PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/00, 13/10, 13/14 — A1

(11) International Publication Number: WO 96/20448

(43) International Publication Date: 4 July 1996 (04.07.96)

(21) International Application Number: PCT/US95/16233

(22) International Filing Date: 22 December 1995 (22.12.95)

(30) Priority Data:
08/362,987 23 December 1994 (23.12.94) US

(71) Applicant: SOUTHWESTERN BELL TECHNOLOGY RESOURCES, INC. [US/US]; 9505 Arboretum Boulevard, Austin, TX 78759 (US).

(72) Inventors: ADAMS, Thomas, L.; 1751 Wild Horse Creek Road, St. Louis, MO 63005 (US). CHORLEY, Will, R.; 251 Ries Road, Ballwin, MO 63021 (US). CUNETTO, Philip, C.; 808 Kentridge Court, Ballwin, MO 63021 (US). DOHERTY, James, M.; 110 Hollybush Court, Ballwin, MO 63021 (US). LEMAY, John, E.; 2161 Hickory Drive, Chesterfield, MO 63005 (US). MUELLER, Stephen, M.; 1348 Autumn Woods Circle, Ballwin, MO 63011 (US). PAROLKAR, Satish; 2315 Westpar Drive, Chesterfield, MO 63017 (US). SCHROEDER, Thimothy, P.; 2142 Terrimill Terrace, Chesterfield, MO 63017 (US). SLATEN, Charles, B.; 337 Forest Grove Court, St. Charles, MO 63304 (US).

(74) Agents: GREENBLUM, Neil, F. et al.; Greenblum & Bernstein, PLC, 1941 Roland Clarke Place, Reston, VA 22091 (US).

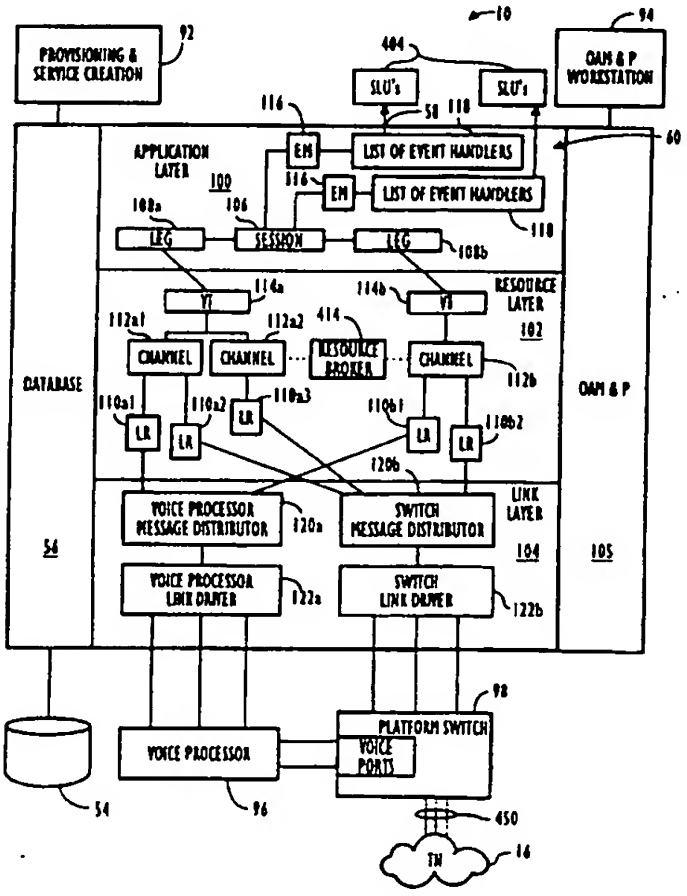
(81) Designated States: AU, CA, JP, KR, NZ, SG, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: FLEXIBLE NETWORK PLATFORM AND CALL PROCESSING SYSTEM

(57) Abstract

A flexible network platform (10a) and call processing system are disclosed. The call processing system includes a particular call processing architecture and a resource managing system. An OAM & P subsystem (94) and systems for performing data provisioning (92) and service creation are each provided. The call processing system may include a call processing mechanism for performing call processing in accordance with defined service logic, and call processing resources connected to the call processing mechanism. The call processing resources are designed so that any service logic unit may be linked to any particular event that might occur in connection with the call processing system. A resource managing system (428) is also disclosed for assigning resources in response to a request made by the call processing system for a specified capability. The resource managing system receives the request for a specified capability and allocates a logical resource object that identifies a particular resource that will support the specified resource capability.



FLEXIBLE NETWORK PLATFORM AND CALL PROCESSING SYSTEM

SUMMARY OF THE INVENTION

1. Field of the Invention

The present invention relates to a telecommunications network and a platform which controls the telecommunications network by managing call processing within the telecommunications network and handling other functions such as the assignment of resources, signalling and switching.

2. Discussion of Background Information

Over the years, telecommunications networks have vastly improved in their ability to enable network users to receive what is termed "personalized services." These services are personalized in that they are oriented toward the user/subscriber instead of the network in general; that is, the user can decide which network service or services he or she wishes to subscribe to, as opposed to the user getting, and paying for, all services that the network provides.

One type of network that is particularly well suited for the provision of personalized services is the intelligent network. Intelligent networks centralize service-related intelligence in special nodes located in the telecommunications network. The intelligent network is well suited for telecommunications systems providing personalized services since it allows the setting up and management of network services that require sizable data and call handling resources.

An article by Sadaba (Telefonica de Espana) entitled "Personal Communications in the Intelligent Network," British Telecommunications Engineering, Vol. 9, August, 1990, pages 80-83, notes the importance of intelligent networks with respect to personalized services. The article generally describes (at pages 82-83) the concept of intelligent networks, and notes that intelligent networks treat all calls individually, dependent upon several parameters and variables. The article further explains that several entities are involved in the control of a call, and that switching functions are clearly separated from the control functions. The article states that separation of the switching functions from the control functions allows the resulting network to be

personalized services, change the resources used by the platform, quickly create and deliver services, and provide OAM&P capabilities.

3. Definitions

The following terms are defined in order to provide a better understanding of the present invention as described herein.

Aggregate resource capability:

A resource capability that represents more than one basic capability.

Alarm factors:

The input states associated with a managed object which caused the object's summary state to take on its current value. If all problems listed in the alarm states are resolved, the managed object's summary state will return to CLEAR.

Application layer:

The application layer comprises all classes whose objects generically participate in call processing, including, e.g., the leg, session, event manager, event handler and application components.

Application component:

A basic unit of service function. Services are built using application components.

Basic capability:

Fundamental units of equipment functionality. A basic capability cannot be decomposed into other capabilities. Some examples of basic capabilities could be playing tones, and recognizing DTMF digits.

Call processing stack:

A hierarchy of objects that interact with each other to implement call processing. From bottom to top for a given active call, the stack comprises one or more logical resources, one or more channels, one or more virtual terminals, one or more legs, a session, and at the top of the hierarchy, a set of event managers, event handlers and application components that are utilized to

Input state:

The set of states which serve as the input to one or more functions used to calculate the output states of a managed object. These states are communicated to the objects via the state distributor.

Intermediate model domain:

Managed object representation of objects existing in the real-time domain (RTD) and logical groupings of those objects.

Leg:

The application layer object that represents a party to a call. The leg manages both information about a single party in a call and the VT (virtual terminal) object associated with that party's call. The leg may also perform other functions.

Link layer:

The software in the platform host that manages the communications link which connects physical resources to the host. The link layer passes messages between physical resources and the logical resource objects with which they are associated.

Logical resource:

An object in the call processing stack that represents one or more resources, e.g. hardware, software, firmware, etc. The logical resource may perform functions such as translating commands from higher layers into bit streams understandable by the resources it represents. Other functions that a logical resource might perform include translating reports from the resources it represents into calls to reporting methods that are present in the higher layers of the software.

Managed object:

A collection of a set of input states, a set of output states, a set of functions which calculate the output states from the input states, and a set of functions which this collected entity can perform. This collection of information within the

The channel with reference to which an event occurs. The reference channel is not necessarily the same as the channel which produced the event. For example, an internal offer connection event is produced by a channel on the incoming side of a call. However, its reference channel is an unspecified channel and some leg on the outgoing side of the call. If a session receives an event with a particular reference leg and a particular reference channel and it contains an event handler for that event, with the particular reference leg and particular reference channel, it starts executing that particular corresponding event handler. The reference channel for an event is contained in a virtual terminal (VT) associated with the reference leg for that same event.

Reference leg:

The leg with reference to which an event occurs. The leg is not necessarily the same as the leg which produced the event (which may be called the originating leg). For example, an internal offer connection event may be produced by a leg on the incoming side of a call. But its reference leg may be some leg on the outgoing side of a call. If a session receives an event with a particular reference leg and reference channel, and it contains an event handler that corresponds to that event, reference leg, and reference channel, it will start executing that event handler. The reference leg for an event is associated with a VT that contains the reference channel for the same event.

Resource capability:

A capability that represents one or more basic capabilities.

Resource layer:

In the context of a flexible network platform, as disclosed herein, the resource layer may comprise all logical resource, channel, and VT (virtual

Summary state:

A state used to describe the state of a managed object. The summary state may include terms such as CLEAR, MINOR, MAJOR, CRITICAL, INITIALIZING or UNKNOWN.

Universal information network (UIN):

A network that deals not only with voice-based personalized services, but handles services handling/using other types of media including audio and video, as well as text information.

Virtual synchrony:

A property of a distributed system under which each recipient of a set of events receives those events in the same order that would be produced if the event happened everywhere at the same time. A virtually synchronous system will not guarantee that distributed events occur at the same time, but only that they will be ordered (i.e., set into motion) at all recipients as if they had occurred synchronously.

Virtual terminal (VT):

An object in the call processing stack that manages a set of channel objects associated with a single user.

Virtual user (VU):

An entry in the database that stores persistent data about a subscriber to platform services. Such data may include, for example, the name, address and subscribed services of that user. Originating callers who have not subscribed to any originating services may be associated with a virtual user entry which corresponds to a default origination service.

SUMMARY OF THE INVENTION

In view of the above, the present invention, through one or more of its various aspects and/or embodiments, is thus presented to accomplish one or more objectives and advantages, such as those noted below.

It is an objective of the present invention to provide

within the flexible network platform, and makes that information easily accessible.

It is yet a further objective of the present invention to provide an OAM&P system that gathers and maintains information about the states of various entities within the call processing mechanism of a platform, without adversely affecting the platform's ability to perform call processing. It is an objective to provide such an OAM&P state information gathering system which does not require the call processing system to take an active part in gathering and maintaining such state information.

It is yet a further objective of the present invention to provide a flexible managed object hierarchy which presents information in a certain standard way to OAM&P information clients (including information about what information is provided, where further information may be obtained, and what actions the clients may take with respect to that particular state information). Thus, it is an objective of the present invention to allow the introduction of new types of managed objects without requiring changes to the architecture or configuration of information clients that may call upon information from the managed objects.

A further objective of the present invention is to provide a flexible network platform which will allow new service provisioning and modification to be quickly and easily delivered. A further objective of the present invention is to provide such a flexible network platform which allows services to execute in an asynchronous environment, but to allow service logic to be programmed in a sequential fashion without the need to handle asynchronous events.

It is a further objective of the present invention to provide service logic that can be interpreted in a threaded fashion in order to allow services to be modified and added to extended data interfaces without the need to recompile or halt the system. A further objective of the present invention is to provide a threaded interpretative service logic mechanism which does not trade speed of execution for flexibility.

It is a further objective of the present invention to

within the initial event handler object. The method of the initial event handler object performs several functions in accordance with variables set forth in the initial event handler object. The variables may include a service logic unit identifying variable, and the several functions may include commencing execution of a service logic unit.

A mechanism may also be provided for registering additional event handler objects in response to requests by application components being executed within a service logic unit. Each additional event handler object corresponds to a service logic unit.

In accordance with another aspect of the present invention, a resource managing system may be provided for assigning resources for use in performing call processing by a call processing system. A resource is assigned in response to a request made by the call processing system for a particular capability. The resource managing system includes a mechanism for receiving a request made by the call processing system for a specified capability. A plurality of logical resource objects are defined, each logical resource object comprising a mechanism for translating generic resource commands made by the call processing system into a form compatible with actual resources coupled to the call processing system. A further mechanism may be provided for correlating which logical resource object supports each capability within a set of capabilities that may be requested by the call processing system. In addition, a mechanism is provided for allocating a logical resource object that supports the specified capability.

The above-listed and other objectives, features and advantages of the present invention will be more fully set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is further described in the detailed description which follows, by reference to the noted plurality of drawings, by way of non-limiting examples of preferred embodiments of the present invention, in which like reference numerals represent similar parts throughout the several views of the drawings, and wherein:

Fig. 15 is a flow chart representing the starting of an origination service once the originating side has been set up;

Fig. 16 is a flow chart representing the setting up of a terminating side of a call processing stack once the originating service has been started in accordance with the process of Fig. 15;

Fig. 17 is a flow chart representing the offering of a connection to a party on the network, after the terminating side has been made in the process of Fig. 16;

Fig. 18A is a flow chart representing the general steps performed in accepting an offered connection;

Fig. 18B is a flow chart representing the steps that are performed in performing an escape service;

Fig. 19 is a flow chart representing the steps that may be performed in completing a terminating side on the call processing software right before a call is actually placed on the network;

Fig. 20 is a block diagram representing objects that are present after a call origination portion of a call has been completed;

Fig. 21 is a block diagram representing the objects present within the call processing software after an outgoing leg has been created;

Fig. 22 is a block diagram representing the various objects that are present within the call processing software after disconnect handlers have been created in relation to the incoming and outgoing legs;

Fig. 23 is a block diagram representing the various objects that are present within the call processing software after an AC (application component) has been executed which creates another disconnect handler for the incoming leg;

Fig. 24 is a general block diagram of the OAM&P portion of a flexible network platform;

Fig. 24A illustrates a managed object;

Fig. 24B illustrates a managed object in relation to its input and output states;

Figs. 25 and 26 are graphic representations of various subsystems of the flexible network platform in relation to the managed object hierarchy of the OAM&P subsystem;

more nodes of another network separate and independent from network 16 illustrated in Fig. 1.

Network platform 10 will typically comprise several subsystems for performing functions including: call processing; database maintenance, control and management; service and user data provisioning; service creation; OAM&P (operations, administration, maintenance and provisioning); and network functions such as routing, translation and switching.

Fig. 2 illustrates a particular example of a network physical architecture within which a flexible network platform, such as the one shown in Fig. 1, can be utilized. In the network physical architecture illustrated in this figure, two flexible network platforms 10a and 10b are provided. A first network platform 10a is connected to two nodes of a telecommunications network, including first and second central office switches 12a, 12b. A second network platform 10b is connected to a node which comprises a cellular switch 14. Each of the nodes specifically shown in Fig. 2 is coupled to a telecommunications network 16 which may, e.g., provide a public switched telephone network (PSTN). Each of the network platforms 10a, 10b is connected to its respective network nodes (central office switches 12a, 12b, and cellular switch 14) via one or more communications links 18, which provide the necessary connection between each network platform and the network node to which it is connected. By way of example, such communications links 18 may comprise digital trunks and/or analog subscriber lines. The present invention does not place any limitation on the types of communications links that extend between network entities and network platform 10. The links can use various types of transmission media including, e.g., two-wire open lines, twisted pair lines, coaxial cable, optical fiber, satellites, terrestrial microwave, and radio transmission. Additionally, if physical lines are used, the types of lines used may be switched connections or permanent lines. The types of communication link protocols may also vary. The diagram shown in Fig. 2 shows a particular implementation of a flexible network platform 10 provided in accordance with the present invention.

lines are terminated directly to the flexible network platforms 10a, 10b. With this physical network architecture, there will be a slightly shorter call setup delay in utilization of services provided by the flexible network platforms 10a, 10b. Such a configuration may be beneficial where the flexible network platforms are providing services only to customers which are based within the associated local exchanges. Difficulties may arise with this configuration in receiving automatic number identification (ANI) on terminating calls that are originating outside of the home local office, since existing trunk signalling to the local exchange does not normally carry ANI information.

Depending upon which services are provided by the flexible network platform, it may be desirable or even necessary to place the flexible network platform outside of the public network. The precise topology of the physical network architecture may vary and need not resemble either of the architectures shown in Figs. 3 and 4.

In order for a call to have access to the services provided by the flexible network platform, a particular service access strategy may be utilized. In this regard, the services may be accessed by addressing the services with the use of dialed numbers (DNs), or, alternately, some other method of addressing. Subscribers may be terminated on the platform or may be hot-lined to the platform. In any event, it may be beneficial to provide some sort of direct user dialing to the platform in order to control calls. For example, direct user dialing to the platform may be facilitated via an integrated voice/data interface, where the data channel is used as a call control signalling channel between the user and the platform.

With the service access strategy utilizing dialing plan access DNs routed to the platform, it may have the ability to address certain components of the flexible network platform, such as a service (independent of a subscriber), a subscriber (providing access to an array of services provisioned for that subscriber), and/or a specific service for a specific subscriber.

By using a dialing plan service access strategy, services

the platform, including, e.g., voice peripheral 26 and data peripheral 28, is shown as being connected to a platform switch 24 via a voice/data path 46, so that any voice, data, or other needed information may be routed to the telecommunications network via the platform switch. Host computer complex 22 has the primary duty of performing call processing functions and executing services in connection with such call processing. Host computer complex 22 calls upon the various peripherals such as the voice peripheral 26 and data peripheral 28 along with the platform switch 24 in order to execute the appropriate services. Provisioning work station 30 may be provided in order to allow entry, retrieval, review and editing of service data as it pertains to the subscribers of the various platform services. Service creation work station 32 may be provided to allow the creation and modification of services delivered to host computer complex 22. OAM&P work station 34 may provide a user interface through which persons can monitor the performance of the components of the flexible network platform. It provides a unified view of the platform's behavior and a common point of control over its operation. Flexible network platform 10 is connected to telecommunications network 16 via communication link 18, which are coupled to platform switch 24.

Platform switch 24, host computer complex 22, and peripheral processing units 26 and 28 together form a core platform system which controls the performance of services. While particular peripheral processing units are illustrated in Fig. 5, including a voice peripheral 26 and a data peripheral 28, other types of peripheral processing units may be provided depending upon the type of functions to be performed by the network platform. In addition, the flexible network platform 10 may include more than one platform switch in order to increase the capacity of switching for the overall network platform 10. In the alternative, other variations may be made to the hardware architecture of the flexible network platform 10, as long as the general functionality of each component remains in the hardware architecture. For example, host computer complex 22 may be expanded to have a platform switch integrally provided therein, and to include an

as shown in Fig. 6, more media processors 50 are provided than necessary; whereas only "n" media processors may be needed at a particular time, an additional number "k" is provided, thus utilizing "n+k" sparing in order to ensure the reliable use of media processors 50. Other redundancies of the hardware architecture include the use of two separate host computers, a first host computer 36a being the hot or active host computer, while the other host computer 36b is in a warm standby state. In addition, two separate power supplies 40a and 40b are provided, and each may be used for either one of the host computers 36a, 36b, in order to provide the necessary redundancy in the event that one of the power supplies fails to operate properly. Another example of reliability achieved by redundancy in the hardware is the use of a dual-ported RAID array 38. In the event that one of the ports of RAID array 38 fails, either host 36a, 36b may use the other port. Other inherently fault tolerance hardware may be utilized. For example, a switch such as the SUMMA 4 switch may be provided which is inherently fault tolerant. The interconnection network 20 may be provided with multiple links between the various devices in order to protect against link failure.

Depending upon the particular environment of the flexible network platform, a NEBS-compliant (Network Equipment Building Standard) physical hardware architecture may be employed in order to implement the platform. This may be required, e.g., if the flexible network platform is going to be employed in a central office environment.

With reference to the example embodiment shown in Fig. 6, it is noted that the dual-ported RAID array 38 may comprise a RAID array provided by Array Technologies. Each of the host computers may, e.g., be implemented with the use of a SPARC 20 host computer. It is noted that reliability concerns revolve around two main issues, including hardware failure and data protection. With regards to data protection, a complete set of the data of the flexible network platform is duplicated within the RAID array 38. Additionally, the set of data is duplicated within the RAM of each host computer 36a, 36b, as represented by mirrored system disks 52a, 52b, as shown in Fig. 6.

switchover mechanism may be implemented by what is called "fail over software" provided within the platform system.

Data protection is another significant issue that will affect the reliability of the platform system. In accordance with a particular embodiment of the present invention, the database of the platform may be stored in an ORACLE database, e.g., a system 7 ORACLE database or other versions thereof. The types of data that may be stored in the database include service data which is provided on a per-user basis, system data (including information such as which media processors are connected to the platform at a particular time), and service logic (in the form of service logic units (SLUs)). The ORACLE system has a transaction mechanism which prevents the corruption of data when it is being placed into the database. When a particular unit of data is being forwarded to the database, either all of the data goes in, or none goes in.

In order to protect the data, another process that may be performed is the periodic backing up of data in the system so that the backup data can be used in the event of data corruption. If there is a breakdown or corruption of data, the uncorrupted data which is stored in a backup storage may be used.

C. The Platform Software Architecture

Fig. 7 is a block diagram which provides a software view of an exemplary flexible network platform 10. The main elements depicted in Fig. 7 include platform switch 24, host 58, ICN 20, voice and data peripherals 26, 28, and OAM work station 34. An outboard database 54, which may be in the form of disk storage, is coupled to the platform host 58. A resident database 56 is maintained within the RAM of host 58, and the data within the resident database corresponds precisely with that of the outboard database. On the software level, platform host 58 includes platform software 60, a distributed process environment 62, and operation system software 64. Similarly, voice processing unit 26 comprises voice software 66, a distributed processing environment 68, and operating system software 70. Data processing unit 28 comprises data software 72, a distributed processing environment 74, and operating system software 76. OAM work

thereby simultaneously handle various tasks. Some example tasks to be implemented or performed by separate processes may include call processing related to the call processing stack; OAM data collection/presentation; communication between various hardware elements; arbitration between active and standby hosts; and logging of events and measurements.

C.1.a The Database

Fig. 9 illustrates a data transformation process that may be performed by the flexible network platform according to the present invention. The particular transformation process shown is an exemplary embodiment. Data that is utilized by the platform software 60 is transformed into database objects 84 through the use of an object interface 88. By creating database objects 84, the data may be stored and later retrieved, thus allowing persistence of the data. The database objects are then transformed via an ORACLE relational SQL interface 90 into database objects in an ORACLE relational database format 86. When the host system is initialized, all database objects are obtained from an outboard database storage. The data being stored in the database includes user data, service logic, and service data. A background function may be provided within platform software 60, so that whenever a change is made to a database object 84, within resident database 56, that same change is written (i.e., persisted) in outboard database storage 54.

C.1.b Distributed Operating System Environment

In accordance with one particular feature of the present invention, the platform may be implemented with the use of a distributed operating system environment. The distributed operating system environment may be created with ISIS. The goal of using a distributed operating system environment is to allow communication between processes on different processors, and to make such communication transparent to the developer as to where the process is running (i.e., across physical boundaries). ISIS is provided with a process reliability mechanism which ensures process reliability by allowing process groups to be called. When a client requests a process group, if a particular process within the process group is faulty, or fails to respond, then ISIS will

logical resource objects 110a, 110b, channels 112a, 112b, and virtual terminals (VTs) 114a, 114b. A logical resource object 110 corresponds to each resource type and is assigned for each resource capability that is to be called upon by the application layer as it executes its service logic. The logical resource objects (otherwise referred to as the LRs) serve to hide the details of the device-specific commands and reports which are generated by the resources themselves and forwarded by the link layer 104 to the resource layer 102 and application layer 100.

A channel object 112 is provided and is formed by a collection of logical resources. Each channel object 112 is associated with a single communications path that is presently connected to platform 10. Channels 112 manage the allocation and release of logical resource objects 110. They also send commands from the higher layers to the logical resources which are provided with the facilities to respond to those commands. A virtual terminal (VT) 114 is formed by a set of one or more channels 112. VTs 114 manage the allocation and release of channels. Taken together, logical resources 110, channels 112, and virtual terminals 114 isolate objects in application layer 100 so that application layer 100 does not have to concern itself with the details of hardware allocation, deallocation, and control. This isolation serves to greatly simplify service programming and minimizes the chance that existing services will be affected when hardware changes are made.

Each session 106 contains a number of event manager objects 116. One event manager object 116 is provided for each type of event session 106 might expect to receive. Events are typically received via legs 108, and handled by an event handler object. Each event manager 116 contains a list of event handler objects 118 set aside to handle unique events.

A session 106 represents a particular active call being processed by the call processing stack. The session 106 groups all event types that it may encounter for the particular active call which it represents. Each event type is represented by an event manager 116 provided within session 106. Each event manager 116 points to event handlers 118

event type provided for within the session object need not be dropped.

When an event handler object is created, e.g., by a leg object or by an application component (during execution of an SLU), the creator of the event handler object communicates with the session object 106, and instructs the session object to register this event handler for this event type. The session will then check to see if there is already an event manager (EM) existing for that event type. If there is an EM already existing for that event type, the session object will instruct the event manager to register the event handler object. The event manager registers the event handler object in accordance with the event handler object's reference channel and reference leg.

Each of the SLUs may be identified by a unique non-zero positive integer, and may be specified as such in one of the variables of each event handler object, so that the event handler object will cause execution of that particular SLU. All event handlers which have completed their interpretation of an SLU are detached from their associated session and destroyed. New event handler objects are created, e.g., by a leg, during some portion of a call setup, or by an application component either during a call setup or during execution of a service. By way of example, if a programming language such as C++ is utilized to create the software that forms the call stack, the various objects, including the event handler objects, may be created by calling a constructor.

When an event occurs, it is reported to the session. If an event occurs and it is an external event, it is reported to the session via the various call components within the resource layer 102, i.e., via a logical resource 110, its associated channel 112, and its associated virtual terminal 114. Virtual terminal 114 then reports the event to session 106 via its associated leg 108. Session 106 will then take the particular event, along with its reference leg and reference channel information, and pass the same to the appropriate event manager 116 for that event type. The event manager 116 will then use the reference leg and reference channel information to find the particular event handler that

session 106. The leg object 108 manages information about the party represented and also controls and manages a virtual terminal object 114 associated in the resource layer 102.

Each session 106 contains a leg list 511 (shown in Fig. 20), which keeps track of the identity of legs 108 associated with a particular session 106. The session 106 also contains an event manager object 116. An event manager 116 is an object in the session 106 that contains the identification of all event handlers 118 for the session 106 and all anticipated events that may possibly occur at any given point during a session. To maintain order, event managers 116 are provided and each event manager handles all events of a given type. The event handler 118 is an object associated with a particular event within event manager 116 that waits for a particular predetermined event to occur, e.g., an external origination event on a predetermined reference channel and reference leg.

If such a predetermined event occurs, the event handler 118 is commanded to start executing a specific service logic unit (SLU or service unit) 404. A service logic unit 404 is a body of logic within the resident database 56 that will be executed upon the occurrence of an event. Conversely, an event is an occurrence during a call that causes the execution of a service logic unit 404. Events are known, predetermined, and anticipated by the session 106. Each event is identified as having a particular reference leg and reference channel. An event can either be an external event or an internal event. An external event is an event generated in telecommunications network 16 external to platform 10. An internal event is an event which is generated by platform software 60. Some examples of external events are: receiving a disconnect signal from the network; failing to receive an answer signal from the network; and an origination signal (trunk seizure) from the network. Some examples of internal events are: offering a connection; and internally instructing a disconnect.

When an event occurs, and the occurrence is communicated to session object 106, session 106 then passes the event along with information identifying the event to an appropriate

creation language may be provided with statements that may be introduced with minimal effort, giving the service programmer access to additional system capabilities as they become available (e.g., voice recognition). This could be done by preventing programming statements from altering the behavior of existing language statements; thus, services which do not require the new capabilities continue to run without change.

A service, one or more SLU's 404 executed cooperatively to provide a set of communications capabilities, is delivered as a single package, and is executed in an asynchronous environment. That is, events occur at arbitrary times and the service logic 404 must be prepared to handle these events without errors, no matter when they occur. Since asynchronous applications are typically more difficult to develop than synchronous ones because of the inability to anticipate when events might occur, the application-oriented language hides this asynchronism within individual language statements. The service programmer may thus write in a sequential programming style, the language ensuring that all relevant synchronous events are handled correctly. This is a considerable simplification for service development.

Service programs executed within the application layer are executed in a threaded interpretive mode. Thus, service programs and their data both appear as data to the application layer 100. This allows services to be modified and/or added through standard data interfaces without the need to recompile or even halt the system. While typical interpretative systems must sacrifice speed of execution for flexibility, the flexible platform of the present invention pays a very small price in speed of execution for the flexibility gained by interpretation of the SLUs. The threaded interpretative mode is employed so that the service program controls execution of a series of compiled modules, application components, which are executed at compiled speed. The sequences of precompiled machine code are what is actually executed giving excellent performance results.

As previously noted, the flexible platform has been designed to maintain flexibility, and therefore, the call stack does not assume a call model, i.e., a template into

one logical resource representing the path, which may be, e.g., a voice path established through platform switch 98. Each channel 112 manages a plurality of logical resources 110, but each logical resource 110 is associated with only one channel 112. However, a single resource may be associated with a plurality of channels as long as a separate LR represents each association with a channel.

Logical resources 110 translate commands from the higher layer objects 106, 108, 114, 112 into physical resource commands understandable by the respective resources they represent. Logical resources 110 also translate reports from the resources represented into calls to the higher layer objects 106, 108, 114, 112.

In addition to these generic call components (including VTs, channels, and LRs), resource layer 102 also includes a resource broker 414 which keeps track of resource types supported by logical resources 110. If a particular resource type is needed on a particular channel 112 in order to provide a resource capability needed for a service, broker 414 will determine whether a logical resource 110 has already been allocated to the particular channel 112 or whether it should be allocated by monitoring the logical resources 110 managed by the channel 112 and determining whether the existing logical resources 110 can accommodate the requested capability.

E.1 Resource Management

Resource layer 102 manages resources by isolating the service logic within application layer 100 from the complexities of resource allocation and deallocation. In order to accomplish this, techniques are used for classifying a resource according to its capabilities, together with mechanisms that use those classifications to present application layer 100 with a view of its platform resources based upon the capabilities of (i.e. the functions performed by) the resources, rather than what specific type of equipment is used to provide a capability. In addition, techniques are used for encapsulating the allocation and deallocation of resources to present application layer 100 with a common, simplified view of the allocation/deallocation process in

100 requests allocation of resources by making a request for one or more capabilities, such as playing tones, recognizing DTMF digits, making or accepting calls. etc. A requested resource capability may be a basic capability which defines a fundamental unit of equipment functionality, or it may be a combination of basic capabilities. LR's 110 may be organized by their channel objects 112 by maintaining two tables keeping track of capabilities. One table keeps track of basic capabilities, and the other keeps track of resource capabilities. The tables may be configured so that each entry in the basic capability table contains a basic capability value and a pointer to an LR supporting the basic capability. Similarly, each entry in the resource capability table may contain a resource capability value and a pointer to an LR supporting the resource capability. As a new LR 110 is allocated to the channel 112, a record is made in each table related to the LR's capabilities and a pointer to its location. Application layer 100 has no direct knowledge of specific LR's 110; it knows only that channels 112 have LR's with certain sets of capabilities. When application layer 100 issues a command to a channel 112, the channel looks in its basic capability table for an LR with a basic capability needed to support the requested action. If no such basic resource is found, a "lack capability" message is sent back to the service. Because the basic capability could not be supported by the channel 112, the service will add the desired functionality to the channel 112 by issuing an "add capability" command directly to channel 112.

Capabilities are added to the architecture because similar pieces of equipment from different vendors do not operate in exactly the same way. For example, a given service A may require both the digitized and synthesized voice resources available from vendor X's voice processing equipment. Assume further, vendor Y makes similar voice processing equipment but only with the digitized voice resource. For service A to operate as desired, it must allocate vendor X's equipment for voice processing. If a service B required only a digitized voice resource, vendor Y's equipment would be allocated to support service B. Thus, in

shown in Fig. 11, which represents the basic capabilities of providing a voice path 430 and playing a tone 432.

The two separate classes of capabilities are desired because some resources support more than one capability. A service may require an LR 110 that supports a particular mix of basic capabilities. In the flexible platform system, this is done by requesting the resource capability that represents a combination of basic capabilities. Thus, this is an efficient way to identify all such combinations without binding service logic into a fixed relationship with a particular resource's configuration. Furthermore, an ordering mechanism, within the resource broker 414, is implemented by mapping out the LR's and the resource capabilities they support so that the resource best suited for a given resource capability may be identified and therefore allocated to the service.

Resource capabilities may also be divided into two subclasses. Figure 11, as mentioned above, shows a block diagram, by way of example, demonstrating the differences between a singular and aggregate resource capability. A singular resource capability represents a single basic capability. Every basic capability has a corresponding single resource capability, for example, bcVoicePath 426 may also referred to as rcVoicePath; bcSynthVox (not shown) may also referred to as rcSynthVox (not shown); etc. An aggregate resource capability represents more than one basic capability, for example, rcBasicVoice 428 represents both rcVoicePlay 430 and rcTonePlay 432.

The resource layer 102 supports several media for the transfer of information from the user to the platform 10, from the platform 10 to a user, or from one user to another user. Each medium is called a path 412, and is characterized by the information carried, not the physical medium on which the information is carried. For example, voice paths carry trunk signalling and voice information; pager paths carry pager signals; and signalling paths carry out-of-band signalling information. An LR 110 associated with a resource that supports a path 412 is known as a path resource 410.

Like the path 412, the channel 112 is an object that

problem is that if the predetermined time is before the end of the call, not all capabilities will be viable candidates for release. Thus, sticky capabilities and partial releases give a way to control what is released and what is retained.

To determine whether an LR 110 may be properly released, the channel 112 must keep track of the basic capabilities and resource capabilities of its associated LR's, as described above in the Resource Management section. It should be remembered that an LR 110 may support more than one resource capability, and that ACs may issue more than one capability request to a channel 112 for a particular capability. This can result in a channel allocating an LR to support several capabilities. Thus, an LR 110 should not be released until every capability it supports has been completely released by the service. To ensure that no capability or LR 110 is prematurely released, the tables within each channel 112 are consulted before release.

E.2 The Resource Broker

As noted earlier, services have no direct knowledge of LR's 110 owned by channels 112. The services know only of capabilities. Therefore, to have resources allocated to a service, the service must present an "add capability" request to the channel 112.

The resource broker 414 keeps track of which resources support which resource capabilities and manages the allocation of the logical resources for objects known as resource owners, i.e., objects (e.g., channels 112) that contain pointers to LRs 110. Figure 12 shows a flow diagram illustrating, by way of example, steps that may be performed by a resource broker upon receiving an "add capability" request from an executing service and allocating a logical resource to a resource owner to satisfy the request. Before a resource capability is allocated, a channel 112 (which serves as a resource owner) receives a capability request from an executing service. Channel 112 calls the resource broker 414 and indicates the resource capability requested with reference to itself. The resource broker 414 tries to obtain a resource 110 which supports the requested resource capability.

The resource broker 414 may comprise a static list (read

of worrying about how to organize and manage resources 110.

Figure 13 shows an example of a simple resource allocation. Assume that a service is operating and (s.1301) requires the ability to both play and recognize voice signals. A (s.1302) request for resource capability is made, e.g., rcPlayRecognize, to (s.1303) the reference channel through which the service is operating. At this point, assume (s.1304) the channel has no logical resource able to perform the desired capability. The resource broker inquires of the requesting channel (s.1305) which logical resources it has previously obtained. The resource broker asks each of these which capabilities they support. "rcPlayRecognize" will not be represented (s.1304), therefore the resource broker continues by looking in its own resource list (s.1306) for a logical resource that supports "rcPlayRecognize." The resource broker finds such a resource, which we will call DigVoxLR (s.1306). The resource broker allocates a DigVoxLR (s.1307). The DigVoxLR, as part of its allocation obtains all hardware resources needed to support it (s.1308). Once all of these have been allocated, the DigVoxLR reports back (s.1309) to the resource broker that its allocation is complete. The resource broker informs the requesting channel (s.1310) that it now has a logical resource that can support rcPlayRecognize and passes the address of DigVoxLR back to the channel.

The resource broker checks the requesting channel for its owned logical resources before allocating a new one to improve the efficiency of hardware allocation on the platform. In the previous example, DigVoxLR might be able to support a DTMF digit collection capability, "rcDtmfCollect," in addition to rcPlayRecognize. If the requesting channel had earlier obtained DigVoxLR in order to support rcDtmfCollect, it would already have a resource that supports rcPlayRecognize. Rather than obtain a second one, the resource broker determines that the DigVoxLR already owned by the requesting channel can be used to support rcPlayRecognize as well.

Another example relieving service designers of managing resources is when a service wants to tear down a channel. The service simply sends a command to release the channel 112

channels 112, VT's 114 and services. For example, VT configurations need not be known by a service ahead of time. A service can start off with a VT, with no associated channels, and it can later add and release channels 112 of any sort, as required by the particular service program. Once a set of channels 112 has been associated within a VT 114, the VT 114 can add and release capabilities through its associated channels 112.

F. An Exemplary Call Walkthrough Procedure

The exemplary platform call processing architecture disclosed herein can be better understood by walking through the platform call processing architecture as a call is placed from an originating party to a terminating party. Figs. 14-18B illustrate several of the main steps the call processing software will perform in handling a call, including setting up an originating side (Fig. 14), starting an originating service (Fig. 15), making a terminating side (Fig. 16), offering connection (Fig. 17), accepting the offered connection (Fig. 18A), and executing an escape service (Fig. 18B). This walkthrough refers to certain events, event handlers, and terminology. However, the specifics referred to in this walkthrough should not be construed as in any way limiting, as variations may be made to various aspects of the call processing architecture without detracting from the flexible and versatile nature of the same.

Fig. 14 is a flow chart representing, by way of example, the establishment of an originating side of a call processing software in response to receipt of an incoming call. A call is made by party A from a telephone to party B (s.1401). The call arrives at the network platform on the platform trunk (s.1402). A platform switch sends an indication of the call's arrival on the platform to the host computer (s.1403). A message distributor associated with the particular platform trunk creates a logical resource (s.1404). The logical resource creates a channel; the channel creates a virtual terminal; the virtual terminal creates a leg; and the leg creates a session.

Once the originating call sets up the originating side of the call, the originating service begins. Fig. 15 is a

default originating service described herein is the creation of a terminating leg to accommodate party B. As shown in Fig. 16, the service creates a terminating leg in step s.1601 and attaches it to the session in step s.1602. The terminating leg checks the subscriber database to identify the receiving party's subscribed services (s.1603) and finds the appropriate service identification (s.1604). The terminating leg creates an event handler for execution upon the occurrence of an internal software indication of offering connection of the originating leg to the terminating leg (s.1605). The receiving party's subscribed service identification is copied into the event handler (s.1606), and the handler is associated with the appropriate event within the event manager (s.1607). This completes the procedure for making the terminating leg.

A next procedural area in the sample call is the method for offering connection from party A to party B. As mentioned above, the offering may be an internal event from the platform software, although that is not absolutely necessary.

Fig. 17 is a flow diagram representing, by way of example, the steps related to offering connection to a terminating party after the terminating leg has been created in the manner set forth in Fig. 16. The originating leg lets the session know that an offer of connection is being made by party A (s.1701). Another event handler, related to a disconnect by the originating party before party B answers, is created by the user or default service currently executing (s.1702). The executing service copies a predetermined service identification into the created handler (s.1703) and assigns the handler to an appropriate event within the event manager (s.1704). The executing service creates another handler (s.1705), this time for an internal disconnect event generated by the platform software on the terminating side of the call, copies a predetermined service identification into the event handler (s.1706), and assigns the event handler to a specific event in the event manager (s.1707). While not necessary, the service identifiers for the external disconnect event and the internal disconnect event may be the same. This is one of the tremendous advantages of the present invention, that service logic is modular and, therefore, reusable for a

of escape digits (s.18B04). If the originating side cannot support digit collection, the channel must communicate with the resource broker to find and allocate an appropriate logical resource to the channel for digit collection (s.18B05). Once allocated, a confirmation indication is sent to the session, and the call proceeds by playing the service message and awaiting the entry of escape digits (s.18B04). Upon the entry of escape digits, the originating party's desired service is performed (s.18B06), e.g., connection to party B.

While the above description was provided to familiarize the reader with a generalized platform routine, the following call walk-through, shown in Figs. 8 and 20-23, will refer to various specific events, event handlers, and service identifications for the purpose of explanation, and should not be construed as in any way limiting.

In the present scenario, an originating phone call is made from non-subscribing party A to subscribing party B. The call from A is routed through a general telephone system until a central office switch routes the call to the platform switch 98 of the flexible platform 10. A call origination report is input to the switch message distributor 120 associated with the line or trunk 450 on which the call arrived. The message distributor 120 creates a logical resource object 502, see Fig. 20, to handle the report, naming it, for example, IncTrkLR. This object represents the hardware of the incoming call to the flexible platform 10, and for the duration of the call, all reports from and commands to the line or trunk 450 go through IncTrkLR 502.

Figure 20 shows a block diagram representing, by way of example, the creation of objects building the sample session of the example from signal arrival on the platform to immediately following call origination. After the creation of IncTrkLR 502, the logical resource object creates a channel object 504, named, for example, IncVChannel. Channel object IncVChannel 504 in turn creates a virtual terminal object 505, for example, IncVT. The IncVT object then creates a leg object 508, for example, IncLeg. Finally, the leg object creates a session object 510, for example, TheSession.

manager 512 will inform event handler EXT_OrigEH 514 to execute the ID 100 service 520 defined therein.

Note that TheSession 510 includes a leg list 511 to keep track of the parties to TheSession's call, which at this time includes only the originating leg, IncLeg. Also note that the event manager 512 is a list of anticipated and predetermined events 506. However, at this point in the call, only one event 506, EXT_orig, is anticipated.

The system at this point will remain idle until an event occurs attempting to establish communication between party A and party B. To get things moving, IncLeg 508, as representative of the calling party, notifies TheSession 510 of the external event, signifying a call from outside of the platform to inside the platform. To process the event, IncLeg passes the external origination event of the call origination, EXT_orig 506, plus identifying itself and IncVChannel as the reference leg 516 and reference channel 518, respectively. TheSession passes the external origination event to the event manager which handles events of this type, and that event manager further passes the event to event handler 514. Since the event 506 is a member of the list, the event manager 512 accesses event handler 514 to respond. Since EXT_OrigEH, for example, is invoked by an external origination event 506, and the call from outside the platform to within the platform occurs over reference leg 516 and reference channel 518, the event manager 512 accesses the event handler 514. The event handler 514 then executes the service logic unit 520 associated with EXT_OrigEH, in this example, the default originating service ID 100.

In a service logic program employed as an originating service by the flexible network platform, a first declared scratch variable, scratch variable 0, may be defined as, for example, called_number. This is important because originating leg 508, as the creator of the event handler, sets the scratch variable 0 in EXT_OrigEH 514 when the object was created, and now every originating service must assume that the leg 508 will set all scratch variables 501 this way.

By way of example, assume:

ID 100

software as a result of the execution of the make_leg AC of the default service. The execution of the make_leg AC creates a new leg called, for example, OtgLeg 608, attaches it to TheSession 510, and assigns predetermined scratch variables associated with a newly created leg. In attaching the OtgLeg 608 to TheSession 510, the session's leg list 511 is updated to include OtgLeg as a party to TheSession. OtgLeg 508 looks in the subscriber database 56 (see Fig. 8) for a record whose key is the same as the value, for example, scratch variable 0, e.g., called_number. Since it has been assumed that party A called party B, a subscriber, a record exists with the value called_number in the subscriber database. The record retrieved is identified as the terminating party's virtual user record and called, for example, OtgVU. OtgVU also contains the service unit ID of called party B's terminating service. For example, ID 10000 could be the service identification number. This is an aspect of the platform's flexible modular code design. Because services are stored according to an identifying number, two services cannot use the same number identifier. However, different event handlers may access the same services, in response to the same or different events, simply by calling the service identification number.

Upon creation, OtgLeg 608 creates an event handler 614, for example, INT_OffConnEH, for an internal offer connection event 606, for example, INT_off_conn. The reference leg 616 for the event handler 614 is defined here as OtgLeg. However, the reference channel 618 is defined as NULL, because no channel has as of yet been created which is associated with OtgLeg 608. This enables event handler 614 to be activated by an internal offer connection event 606 no matter what reference channel it is passed through, so long as the reference leg 608 is OtgLeg. The offer connection event 606 is considered an internal event because the occurrence comes from the flexible platform software, not from some device outside of the flexible platform.

OtgLeg 608, the terminating leg, copies ID 10000 service 620 into event handler INT_OffConnEH 614 and registers the event handler 614 with TheSession 510. Note that the

The ID 1004 service 720 is executed upon the occurrence of an external disconnect event 706. Because this particular service is a background service, the offer_connection AC must initialize the event handler's scratch variables 701. For example, scratch variable 0 is set to the address of the AC to be called when the invoke statement of ID 1004 service 720 is executed by the handler 714, e.g., the leg class's static ac_entry method; scratch variable 1 is set to the ID of an AC to be executed upon leaving the background service 720, e.g., olc_e_disc AC, an offer leg connection AC that handles an external disconnect on behalf of the offer_connection AC; scratch variable 2 is set to the address of an object that should execute the AC whose ID appears in the invoke statement, e.g., IncLeg; and after the scratch variables of the background service have been initialized, the offer_connection AC sets background parameter scratch variables specific to the olc_e_disc AC, e.g., scratch variable 3 is set to the address of EXT_OrigEH 514, the event handler that created EXT_DiscEH_{Inc} 714.

The AC offer_connection creates another event handler 1714 that responds if and when an internal disconnect event 1706, e.g., INT_disc, is generated by the software with reference to any channel in reference leg 1716, e.g., OtgLeg. This handler 1714 is referred to, e.g., as INT_DiscEH_{Otg}. The service ID 1720 assigned by the offer_connection AC to the handler 1714 is, for example, ID 1004, the same service associated with EXT_DiscEH_{Inc} 714. As alluded to above, the service identifiers for the external disconnect event and the internal disconnect event may be the same. This reusable code in the software enables the service designer to associate any event with any service logic, thus, making the service more flexible and service design easier. This handler is also registered with TheSession 510. Note that an external disconnect handler has not been created for the outgoing side because there is no connection, at this time, to the outgoing side that might produce such an event. As noted above, the ID 1004 service 720, 1720 is a background service that requires the creating AC to initialize the service scratch variables 701, 1701. The initialization is the same, except

```
        proceedToNext.
    initialization AC
        initialize number to call;
        proceedToNext.
    play_message AC
        allocate resources for service support;
        proceedToNext.
    mlr AC
        call number to call;
        proceedToNext.
```

The ID 10000 service 620 begins with, for example, a record_time AC, which records the current time into a scratch variable. This variable can be employed as a reference point for billing and other vital statistics, etc.

Because the outgoing side has accepted the offered connection through TheSession's activation of the appropriate event handler for an offer of connection, it would be possible for the outgoing side to generate an internal disconnect event 1706 within a reference leg and reference channel, e.g., IncLeg and IncVChannel, respectively. The record_time AC's proceedToNext statement causes INT_OffConnEH 714, for example, to reg_for_i_disc AC, an AC that registers the originating leg and channel as the reference leg 1716 and channel 1718 for internal disconnect events 1706. Figure 23 shows a flow diagram representing, by way of example, the objects of the call processing software following the execution of the reg_for_i_disc AC. The figure shows that this AC creates an event handler 814 for executing a given SLU upon occurrence of an internal disconnect by the platform software on the originating side of the call, for example, INT_DiscEH_{Inc} 814, and registers it with TheSession 510. The new event handler 814 is set to execute, for example, the ID 1005 background service 820. The handler 814 must deal with internal disconnect events 1706 within reference leg 816 and reference channel 818, e.g., IncLeg and IncVChannel. The scratch variables 801 for INT_DiscEH_{Inc} 814 must be initialized because service ID 1005 820 is another background service. For example, scratch variable 0 is set to the address of the event handler class's static ac_entry method; scratch variable 1 is set to the ID of the tear_down AC, the AC that tears down one leg of a disconnected call; scratch variable 2 is set to the address of the object that will execute the tear_down AC,

required to produce and collect DTMF digits are available and ready on the originating channel. In this configuration, for example, the digit collecting capability is supported by the same hardware that supports the capability for playing voice messages, therefore, the digit collecting capability is obtained by playing a null message via, e.g., a play_msg AC on the originating channel. The originating party does not hear the null message, but the necessary resources are obtained and made ready to use for both voice message playing and digit collection. The resource layer allocates a logical resource suitable to perform the capability in a manner commensurate with that set forth above in the section describing the resource manager. Thus, if the hardware that supports playing voice messages did not support digit collection, another logical resource object would be appended to IncVChannel through the resource broker 414 (see Fig. 8). Another alternative is to initialize the system through a software setup macro with an appropriate code.

Finally, the proceedToNext statement within the play_msg AC causes INT_OffConnEH 614 to execute, e.g., mlr AC, represented by a call statement within the program for the ID 10000 service 620. The mlr AC has parameters including, for example, the number to be called; the leg on which the call is to be made, e.g., the reference leg for the service, e.g., OtgLeg; the leg and channel monitored for escape while the call is being made, e.g., the reference leg and channel of the originating call for the service, i.e., IncLeg and IncVChannel; and a set of three parameters that are updated if and when the originating call is answered, e.g., ans_channel set to the ID of the channel in the reference leg that answers, ans_number set to the network address of the station represented by ans_channel, and ans_type set to a character code, determined by the service itself, used to convey information about the characteristics of the answering station. While these parameters may appear to complicate a simple phone call, it should be remembered that the mlr AC can make a call to more than one network address at the same time. Thus, it is important to know precisely which station has answered and where to find its channel.

the time needed to develop a new provisioning application, it is desired that various parts of the provisioning application be standardized so that it may be re-used. Accordingly, it is desired that the communications code be re-used, i.e., standardized. In addition, it would be beneficial if the data model component of the provisioning application could be standardized so it could be re-used. It is not an easy task to standardize the data model component, since the data model typically needs to be changed for each service. For example, in prior provisioning application systems, if one service needs paging information and another service does not need paging information, the data model will need to be changed. In any event, even if the data model and the communications component were configured so they would not need to be changed for each new provisioning application, there still need to be changes in the user interface.

In one aspect of the present invention, a provisioning application base is provided, which comprises both a standardized data model component and a standardized communications component. Fig. 27 shows the relationship between a provisioning system 132 and a service platform 130. Service platform 130 is shown as having service logic 134, and a service data model having data objects 136. Service platform 130 is coupled to provisioning application 132 via a transmission medium 138, which may be any appropriate transmission medium. User interface software 140 is provided within provisioning system 132, and is connected to a data model component 142, as well a communications component 144. User interface 140 is coupled to data model 142 and communications component 144 via an applications program interface (API) 148. API 148, data model 142 and communications component 144 together form what is known as a provisioning application base 146, i.e., portions of the provisioning system which have a standardized structure and can be re-used when introducing new provisioning applications.

The provisioning application base is standardized because it does not have any service-specific models built into it, and can hence support provisioning applications for any services.

data-object basis, and does not need to respond in any unique manner to different combinations of data object structures in accordance with a particular service. Accordingly, data model 142 can be standardized so that it contains no service-specific knowledge, and requires changes only when a new data object is introduced or an existing data object is modified.

User interface 140 allows service specific applications to be developed for each service. Such service specific applications can be developed using any development environment that is appropriate for the target platform and that is capable of making API calls to provisioning base 146.

User interface 140 should be configured so that it knows which data objects are needed for a particular service provided by platform 130. For example, suppose there is data within platform 130 that is stored in accordance with a particular service data model, and provisioning system 132 is to be used to change that data in accordance with a provisioning application that already has been built. Since user interface 140 already knows which data objects need to be pulled from platform 130, it can call such data objects by using a function call utilizing keys for the objects that it wants, each key indicating the particular object that it wants. An API call is made to communications component 144 to send a request to platform 130 with a list of keys to particular data objects that are wanted. The platform 130 will look up and retrieve the appropriate data objects and send them back via transmission medium 138 in the form of a provisioning protocol representation.

The provisioning protocol allows the data objects to be easily sent over transmission medium 138. The provisioning protocol entails encoding, in ASCII format, the data objects and transferring them in a serial fashion.

Provisioning system 132 then reads the protocol utilizing communication components 144, and recreates the objects in memory on the provisioning side.

In the provisioning system illustrated in Fig. 27, there are only a fixed number of data types which have a certain defined structure. In addition, the data model component deals with each data type in the same way each time a data

object may include information such as user ID, graph ID and data number. The key for a particular virtual user object may be indicated by a user ID. The keys are used when a provisioning system 132 sends and retrieves data to and from platform 130.

When provisioning base 146 retrieves objects from platform 130, it does so at the request of user interface 140. Typically user interface 140 will request that all data objects corresponding to a particular service and user be retrieved at one time. When those data objects are returned to the provisioning base 146, they are available to user interface 140 for display and editing.

Each service data type/data object may be formed with a hierarchal tree structure so that the data organization in the provisioning base 146 which forms data model component 142 can be thought of as a forest of object trees. Each root object is associated with its key (which may also be referred to as a label). User interface 140 can obtain and set data within data objects by referencing them in API calls referring to their labels.

API function calls may be provided in four general categories, including configuration, communications, access and data positioning. Configuration API calls are used to set provisioning base parameters that affect subsequent API calls. Communications API functions are used to tell provisioning base 146 to send or retrieve data to or from a platform 130. Access functions are used by user interface 140 to get and set data in data model component 142, and thus are used to get and set data without the need to transmit data between provisioning system 132 and platform 130. Data positioning functions are used to set the context for subsequent API calls.

User interface 140 may reference objects in the provisioning base data model component 142 in many different ways. In accordance with one particular referencing mechanism, provisioning base 146 may keep a pointer (referred to as a DP) to a "current" data object. Access and data positioning API calls may implicitly act on the object pointed to by DP. For example, if DP points to a screening object,

is the user ID.

After issuing a retrieve operation, assuming the retrieve was successful, user interface 140 may assume that the data now resides in the data model components 142 and is available for access via API 148. User interface 140 may now use API 148 to request data strings necessary to fill its lists, text boxes and check boxes. When this is completed, user interface 140 may relinquish control and wait for user input. After changes are made to the data, user interface 140 may make API calls in order to update data model component 142.

Eventually, a send operation will be requested by the provisioner, at which time provisioning base 146 will determine which objects have been modified, and will create and send a protocol string to platform 130 in order to update the modified data within platform 130.

If data for a particular subscriber that is new to the service is created, the provisioning application may use API 148 in order to clear out the existing data in the provisioning base data model component 142, and build "from scratch" using API 148 the minimum data objects trees required by the service and send them to platform 130. This must be done because platform 130 does not store any form or template for a particular service.

User interface 140 may be designed in a manner different to that described above. For example, user interface 140 may be designed so that the service provider can perform service provisioning together with data management. This would be allowed by provisioning base 146 since it is generic enough to allow any number of a variety of application views into the platform data. Moreover, provisioning base 146 does not have to be the only layer below the application software. The application, which includes user interface 140, may manage data, some of which is held within platform 130, and some of which may be held in other databases. The application software may be designed to present an integrated view, and interact with the appropriate databases as needed.

Figs. 28A-28K illustrate various types of data objects that may together form a data model for a platform 130 such as that shown in Fig. 27. The various data objects

is performed, user interface 140 preferably does not store its own copy of the data, modify the same as the user edits the data, and then sends the revised data back to the data repository. Rather, user interface 140 will modify data within provisioning base 146 directly, as the user makes changes on the screen. If certain data is not modified at all, it is only used to fill in the fields of user interface 140 and is then discarded. That is, non-changed data will not be re-sent to the data repository within provisioning base 146 when the data is returned on execution of a get call.

Fig. 28A illustrates a case database object 204 which is used for storing case labels utilized by service logic units of the network platform of the present invention. Case database objects 204 include the following fields: case type, value, and completion code. Each case database object 204 comprises one or more case entries 226(1) - 226(N). Each case entry comprises a plurality of nodes/fields, including a case type field 228(n), a value field 230(n), and a completion code field 232(n). The case type field may include information in the form of a short that specifies how the contents of the value field 230(n) are to be interpreted. A case type field 228(n) may include a numerical value, e.g., 1 representing a long, and 2 representing a network address. Each value field 230(n) is a character containing a string representation of the case label to which this record corresponds. By way of example, the character string "123" represents the case label 123. The completion code field 232(n) may be a short whose value is the completion code that is to be returned when a parameter that matches the value within the value field is passed to an AC that has been passed this particular case database object 204 as an entry in a case data list object. Each of the dashed rectangles A1-A3 indicates sets of data items that may be accessed via a single API get call when a data pointer is directed to the root node of case database object 204. Dashed rectangle A1 represents a get entry list call which causes the complete list to be retrieved for use by user interface 140, including the case type, value type, and completion code for each case entry 226(1)-226(N) of case database object 204. Dashed rectangle A2 represents a get

containing a string representation of data (e.g., "123" represents the integer 123).

The initialize database object 208 may be stored as a sparse array. Unlike the case database object 204 (where each item must have the same type), all of the initialize entries 242(n) of an initialize database object 208 need not be of the same type. It is noted that each of the value fields 248(n), represented by a rectangle having rounded corners, may serve as a root node for another data object, including another initialize database object 204. A dashed rectangle C1 is shown around each of the value fields 248(n), and represents an API get call for getting the value list. This API get call may be limited so that it is valid only if all of the initialize entries have the same type as represented in the type field 246(n), and if that type is either a long, a network address, or a character string.

Fig. 28D illustrates an MLR database object 210. A plurality of MLR entries 250(1)-250(N) are provided within the MLR database object 210. Each MLR entry 250(n) has a plurality of fields, including a network address field 252(n), a ring length field 254(n), and an MLR type field 256(n). Each network address field 252(n) is a string containing a directory number called by a service corresponding to the MLR object or MLR service component (i.e., AC). A ring length field 254(n) is a short representing the amount of time the network address is to be rung before assuming that there is no answer. An MLR type field 256(n) is a character field that may be used for any purpose.

A service component (AC) that may accept a database pointer to an MLR database object 210 may be the MLR AC, which simultaneously offers calls to all network addresses in the MLR database object 210. If the MLR list within the MLR database object 210 is empty, the MLR AC will exit with a completion code "MLR no answer."

Fig. 28E illustrates a play announce database object 212, having a plurality (1-N) of message ID fields 258(n). All the message ID fields 258(n) may be provided to the user interface 140 of provisioning application by using an API get call represented by a dashed rectangle E1 as shown in Fig. 28E.

by dashed rectangle G1 which represents an API get call that may be called by user interface 140 of the provisioning system. The complete list of message IDs 266(n) may be called by an API get call as indicated by the dashed rectangle G2.

Fig. 28H illustrates a record message database object 218. This database object includes a time out max field 276, a time out silent field 278, a time out tail field 280, and an eorc symbol field 282. Time out max field 276 may be a short whose value is the amount of time allowed for a voice message recording session. Time out silent field 278 may be a short whose value is the amount of time allowed for a silent period after a recording session is activated. Time out tail field 280 may be a short whose value is the amount of time allowed for a silent period after a voice message or part of a voice message is recorded and before expiration of the value indicated in the time out max field. The eorc symbol field 282 is a character whose value is a key pad character (e.g., #) that may be entered to terminate a voice recording before expiration of the amount of time indicated in the time out max field 276.

Each of these fields may be called together by using an API get call as indicated by dashed rectangle H1.

Fig. 28I illustrates a screening database object 220. This database object includes a plurality of network address fields 284 which may be in the form of an array. Each network address field 284(n) may be a string containing a directory number against which calls are to be screened. A dashed rectangle I1 represents that an API get call may be utilized to provide the data for all of the network address fields 284(1)-284(N) of a particular screening database object 220 to the user interface 140 of the provisioning system 132 as shown in Fig. 27.

Fig. 28J illustrates a time of day database object 222. Each time of day database object 222 may have a plurality of time of day entries 286(1)-286(N). Each time of day entry 286(n) may be provided with a plurality of child nodes/fields, including a start day field 288(n), a start hour field 290(n), a start minute field 292(n), an end day field 294(n), an end hour field 296(n), an end minute field 298(n), and a time

switch 322, a voice processing unit 324, a graphic user interface (GUI) 316, and a plurality of operations support systems 318a, 318b. Call processing system 320 may be provided within the same host computer as the platform OAM&P software system 330.

Platform OAM&P software 330 is shown as comprising a state distributor 308, a presentation domain 310, an intermediate model domain 312, and a real-time domain 314. Presentation domain 310, intermediate model domain 312, and real-time domain 314 together form a managed object information engine. The managed object information engine is produced, in part, by a managed object compiler 342. OAM&P subsystem 330 has three different execution domains, which are indicated as presentation domain 310, intermediate model domain 312, and real-time domain 314. Real-time domain 314 comprises those components of the system which directly support call processing, indicated by call processing system 320, as well as OAM&P specific object which interact in real time with call control objects which form part of call processing system 320. Such OAM&P specific objects may comprise external processes called "cloud" processes such as the processor monitor process 332, which monitors the state of call processing system 320, a switch device process 334 which keeps track of the state of switch 322, and a voice processor device state process 336 which keeps track of the state of voice processing units 324.

Presentation domain 310 comprises applications that are used to present the objects in the intermediate model domain 312 to an external system, which may be a graphical user interface 316, or an operations support system 318a, 318b, e.g., a Bellcore NMA system.

Intermediate model domain 312 comprises managed objects 326 and transient objects 328, which together form a managed object hierarchy. Managed objects 326 (as well as transient objects 328) comprise object representations of objects existing in the real-time domain 314 and logical groupings of those objects.

State distributor 308 is a communication mechanism for simultaneously informing multiple processes of changes in the

standing for physical class), of a particular instance (* indicating all instances of that particular class). The type of state information, i.e., the type of event being registered for, includes the summary state information. Accordingly, when a transition is registered for by a particular process, the process will indicate the process group, its class/type identifier, its instance identifier, and its name (i.e., type of event being registered for).

The command rT(*,*,*,summary) will register the present process to receive all of the summary states from all of the classes/types within each process group and instances of those classes/types. In this manner, all of the summary state information of all the managed objects will be presented to the process that has registered that transition.

The system may be implemented so that there is no persistence implemented in state distributor mechanism 308. When a particular state information server or state information client process terminates, all the registrations and current state information will be removed. This information needs to be rebuilt when the platform is booted up.

The presentation domain contains the application software including graphic user interface (GUI) software 338, and OSS interface application software 340a, 340b, which is used to present the objects that are present in intermediate model domain 312 to an external system, such as a GUI system 215 or an OSS system 318a, 318b.

OAM&P subsystem 330 is formed with a managed object hierarchy. All of the managed objects 326, 328 which are present in intermediate model domain 312 share certain key characteristics which enable them to create an extremely flexible model. An object compiler 342 is provided for creating executable code for each managed object in order to allow a developer to concentrate on the managed objects rather than the environment in which they are created. State distributor 308 is used for inter-object communication, and thus provides distribution capabilities as well as "virtually synchronous" updates to all objects using a particular state, i.e., registered to receive a particular state as a state

a component of the system corresponding to that managed object, and rebooting the complete platform system.

Each managed object may be configured so that it has a distinguished output state known as a summary state. This state may be used to describe the overall state of the managed object, e.g., information indicating the managed object as CLEAR, MINOR, MAJOR, CRITICAL, INITIALIZING, or UNKNOWN. Other information that may be included with the summary state is a list of "alarm factors." The alarm factors are the input states which caused the summary state to take on its current value. For example, if the input states of a certain object are file system 1, file system 2, and file system 3, and the state of file system 1 is 0% available, while the states of file system 2 and file system 3 are each 40% available, the summary state of the managed object may be MAJOR. The alarm factors would include file system 1 only, because file system 2 and file system 3 are not contributing to the summary state being MAJOR rather than CLEAR.

Each time the summary state of a managed object changes, it may be logged into a system log. By doing this, the system log can be searched to determine the state of the managed objects at a previous time.

Fig. 24 illustrates a particular example managed object hierarchy of managed objects 326 and 328. A root managed object 354 represents the overall platform, and has input states which are forwarded from the output states of managed objects corresponding to the resource group 356, voice resources 358, physical processor 360, and a logical processor 362.

Logical processor 362 has as its input state an output state of a processor monitor process 332 which is in real-time domain 314. Physical processor managed object 360 has input states which comprise an output state of a RAID managed object 364, and an output state of a voice processors managed object 366. RAID managed object 364 has input states which comprise output states of managed objects representing several RAID subsystems and components including a RAID disk managed object 368 and a RAID fan managed object 370.

Voice processors managed object 366 has as its input

process which monitors the activity of the voice processing units 324 is shown as indicating a current error state. That is detected by voice processor 1 managed object 366 since it has registered a transition as an information client for that particular state (or state information including such an error state). That information can be determined by a user using one of OAM&P workstations 319 in a number of ways.

An error condition is detected by a monitoring process (not shown) which monitors one or more states of voice processing units 324. That error condition is reported to the OAM&P sub-system by generating (i.e., reporting) a transition to the state distributor. Voice processor 1 object module 366 is then notified of the error state, since it has registered to receive the state of voice processing unit 324 via the state distributor. The output state of voice processor 1 managed object 366 will then change according to the change in its input state. That output state forms an input state of voice resources managed object 364, and accordingly will change its output state, which will in turn cause call processing resources managed object 358 and platform managed object 355 to change their respective output states. Accordingly, OAM&P work stations 319 can detect a change in the output state of platform managed object 355, indicating an alarm condition due to there being an error in voice processing units 324.

An OAM&P work station 319 may also directly view the attributes of a particular managed object. Therefore, OAM&P work station 319 may ask voice processor 1 managed object 366 about its attributes, sub-components, and functions that it can perform. The attributes of that managed object will include the output states, the input states, and alarm factor for that managed object. Accordingly, voice processor 1 managed object 366 may be viewed directly in order to determine that there is an error within voice processing units 324. That managed object may then have functions it can perform or that can be directly invoked by an OAM&P work station 319. For example, OAM&P work station 319 may desire to switch from one voice processing unit which is faulty to another voice processing unit. It may be one of the functions

abnormal termination.

Transient objects are useful where there is uncertainty in the number of components of a system. A particular flexible platform may be equipped differently with network connections, voice processors, etc., and thus cannot be prepared to handle any type or number of components connected to the platform system. In order to provide OAM&P state information for such items which will vary in number, the managed object hierarchy may be modified at each system installation, by developing specialized static managed objects that represent different components, or a single managed object may be created which can represent multiple components as a single managed object. Another way of handling such a situation is with the use of transient objects which may be created by a parent managed object.

In other words, the flexible network platform may have more than one instance of a hardware type. For example, there may be a plurality of media processors that are used as peripheral processing units, and that number may vary from platform to platform. A transient object template may be created which has information which pertains to the transient object that could be created for each instance, including output states, input states, a list of functions the transient object can perform and logic indicating what effect the input states will have on the output states of the transient object. In addition, the transient object template will have a transient create function associated therewith. A monitor process monitors the media processors and therefore knows the number of media processors which are connected. Accordingly, that media processor monitor process will look at the hardware and decide how many media processors are connected and invoke a create transient object function for each media processor for which a transient object is needed. The managed object hierarchy shown in intermediate model domain 312 in Fig. 24 illustrates a transient object template 328a, as well as transient objects 328b and 328c which were created using transient object template 328a. Each transient object corresponds to a particular voice processing unit among voice processing units 324 that are connected to the platform.

these elements may be preferable.

A special service creation environment will preferably comprise hierarchal diagrams, since as service logic diagrams grow, they become increasingly difficult to comprehend unless there is a mechanism to represent them in hierarchal form. In order to achieve the goal of creating a program representation that allows quick assimilation of the logic, the programming tool may be configured so that it supports hierarchal representations, along with a powerful navigation capability to move through the hierarchy while mentally following the execution path. It may further be preferred or desirable to incorporate a service provisioning function within the service creation environment since the service logic will define the data relationship that will be needed for the created services.

The flexible network platform disclosed herein may write service programs in a specialized application-oriented language. The primitive statements of the language are called application components (ACs). The flexible network platform may be configured so that it executes a machine readable version of its program. A service creation environment may be provided which allows service programs to be expressed in human readable format and to be expressed mainly graphically, rather than in texts. Programming consists of drawing, as well as typing (at lower levels). The resulting program is called a service graph. That graph is then "compiled" in the service creation environment, and the resulting machine-readable version of the program may be downloaded to the flexible network platform where it can be executed.

Because the programming environment may be tailored to a particular application domain, not only is the language customized, but specialized data objects may exist that all programs interact with. These include a session, which represents the current call, and leg, which represents users on the call. These and other data objects are visible in the service program of the flexible network platform, and the service creation environment may create programs utilizing these objects.

A service graph may be provided within a service creation

AC via environment or scratch variables. An AC may use its parameters for input, output or for both input and output purposes.

In order to program with the graph system, a user will lay out the particular AC nodes to execute and interconnect them with edges that specify the sequence of the execution, and the user will then specify the AC parameters on every edge. The service creation environment should support these functions as well as support the programmer in managing the environment variables and scratch variables. The service creation environment may include standard file manipulation capabilities, such as saving, loading and printing a draft. An editor may be provided in order to allow manipulation of source files, which are in the form of graph diagrams. Such editing capabilities may include graph navigation abilities to particular nodes in the graph, the ability to cut and paste portions of graph and other editing functions.

A specific embodiment of a service creation environment will now be described in relation to the particular embodiment of a flexible network platform disclosed herein. The service creation environment may be implemented on an Apple Macintosh in a Smalltalk programming environment. More specifically, the Macintosh may comprise a Macintosh IIcx, or a IIfx, with a 19-inch monitor. The Smalltalk programming environment may comprise Digitalk's Smalltalk VMac Ver 1.2.

The service creation environment may be configured so that it allows a basic graph structure to be built by point and click techniques. In addition to nodes and edges, a number of other objects may be placed on the graph to aid the programmer in producing a graph that is easy to understand, and such additional objects will be described below.

In order to display graphs in a hierarchal fashion, most graphs will take the form of a tree. The top level is referred to as the main graph, or the top level graph. Besides nodes and edges, the top level graph can contain objects (called expanders) which represent hidden (sometimes collapsed) subgraphs. Those expanders can be opened to display the subgraph they contain. The subgraph may in turn contain expanders, with no limits as to the degree of nesting

Each edge 906 will have a "from number" 908 representing the number of the node from which the edge is coming, and a completion code 910 which represents the completion code of the node from which the edge is coming that will trigger the transition that present edge 906 represents. Edge 906 also displays a number representing the node to which the edge is pointing by a "to number" 912. Edge label 907 will optionally include node parameters that the program has specified. Until all required parameters have been specified, the edge label may be provided with an indicator to indicate that this is the case, e.g., by placing a dark underline (not shown) underneath edge label 907.

Comments may be used within a graph and are indicated by free form text. Any object can have a comment except for a connector and a joint graph objects. The service creation environment may be designed so that comments can be placed and moved anywhere on the graph, and so that comments are considered relative to the object that it is associated with. If an object is moved, its comments may automatically move to maintain the same relative position with respect to the object. Comments may automatically be projected back onto the graph if their new relative position would have moved them off the graph.

Fig. 29C illustrates a connector graph object 916 which may represent other nodes in the graph. Connectors 916 allow the graph to be displayed more cleanly by eliminating the need for edges that span a large distance in the graph. Edges can be constructed to connectors but not from connectors. Connectors cannot be created for nodes in other subgraphs, only for nodes in the current subgraph. This is because connectors are not for the purpose of creating edges that span subgraphs, but merely simplify the display of a single subgraph. Each connector may include a connector type 918 (which by definition is equal to the type of the node represented by this connector), and a connector number 920 (which by definition is equal to the type of the node represented by this connector).

Fig. 29D illustrates an expander 922 which is a single object which represents complete subgraphs hidden "below"

a window 952 as shown in Fig. 32 to be opened to display information about that node. The window illustrated in Fig. 32 corresponds to a "play_collect" node. Most of the fields provided in this window are informational, and no input should be required by the programmer. A notes field is provided which can be used for any detailed comments as desired. If this node uses data from a database (not parameters) this field can be used to document the data and its relationship to data used by other nodes. This information can then be made available to a provisioning application designer.

An edge may be opened, e.g., by opening an edge window 954 as shown in Fig. 33. In this window, parameters to ACs may be specified. Therefore, an edge window 954 will most likely always eventually be opened, and user information will be input by that window. The service creation environment may be configured so that it automatically opens an edge window whenever an edge is created.

Mnemonics 999 for the parameters of the "to" node appear on the left side of the parameter list edit box 956. Variables are available as parameters appear in a graph variables list box 958 which is shown at a right side of edge window 954. The service creation environment may be configured so that clicking on the variables in graph variables box 958 will copy a variable's name to the highlighted parameter slot 957 in parameter list box 956, and increment the present highlighted parameter to the next parameter slot below the highlighted parameter slot 957. In this edge window 954, the point-and-click style of programming is shown as being heavily used. The programmer does not have to re-type variable names. The mnemonics 999 on the left of the parameter list box 956 make it clear how many parameters are required.

Other graph objects may be opened in order to open up different types of information. For example, an expander may be opened to thereby open a graph window on the subgraph of the expander. A comment may be opened to open a text edit window on the comment's text. A ghost object may be opened thereby opening the graph of the node of the "other" end of the edge represented by the ghost. A ghost, as noted above,

placed in the menu of legal variable names to chose from.

By providing a data dictionary window, this would allow the programmer to create, delete, or change an alias name. Changing an alias name may result in a global substitution of the new name for the old name on all edges where the old name is used as a parameter. Deleting an alias name could cause edges that have that variable as a parameter to become incomplete. Accordingly, the programmer should be warned before deleting this alias. A finding mechanism (if provided) may be used to find those edges that were rendered incomplete as a result of the deletion.

Routing and Translation

The resource broker may be provided with a mechanism for performing routing and translation. Translation consists of translating an address to particular equipment and routing consists of determining a particular location for the equipment. A given address is routed and translated to provide a particular type of equipment at a particular location.

There are many different ways in which routing and translation can be performed, and it is not necessary that the routing and translation mechanism of the flexible network platform be provided within the resource broker.

The present disclosure describes many different objects in the context of an exemplary embodiment platform software architecture that may be provided in a flexible network platform. In describing the objects, which include objects such as session objects and logic resource objects, specific definitions are provided regarding the encapsulation and modularity of each object. Those definitions are provided to illustrate a specific implementation of the platform software architecture, but they are not intended to be rigidly interpreted. The present invention does not preclude the use of other similar objects that may happen to have different encapsulation and modularity characteristics.

Further, while the invention has been described with reference to preferred embodiments, it is understood that the words which have been used herein are words of description, rather than words of limitation. Changes may be made, within

WHAT IS CLAIMED IS:

1. A telecommunications services network platform for controlling the processing of calls in accordance with one of a plurality of defined services, said network platform comprising:

a call processing system for performing call processing in accordance with defined service logic; and

call processing resources connected to said call processing system;

said resources comprising:

at least one media processor; and

a switching system, connected to a telecommunications network, to said call processing system and to said resources for routing information among said at least one media processor and entities connected to said telecommunications network;

said call processing system comprising:

session means for representing an active call with a session object;

means for forwarding external events associated with said active call to said session object;

means for creating an initial event handler object by specifying a particular method to be called in relation to said initial event handler object, several variables, and values to be placed within the variables, and means for registering said initial event handler object with said session object, said initial event handler object being associated with a particular event which is identified by an event type and a communication path associated with the particular event;

said session object comprising means for receiving the particular event and for calling up said particular method within said initial event handler object;

said platform further comprising means for carrying out said particular method within said initial event handler object and for performing functions in accordance with the variables of said initial event handler object, the variables of said initial event handler object including a service logic unit identifying variable, said functions including commencing

10. The telecommunications services network platform according to claim 7, wherein each said service logic unit comprises a plurality of application components, each application component identifying a pre-compiled procedure that can be executed in a threaded interpretive fashion by an event handler assigned to the service logic unit.

11. The telecommunications services network platform according to claim 1, further comprising an OAM&P system for gathering and maintaining information about the states of entities within said platform, said entities comprising one or more of said resources and one or more objects within said call processing system, said OAM&P system comprising:

a hierarchy of managed objects, each managed object having input states, output states and a defined logical relation among said input states and said output states, managed objects, which have output states that form the input states of other managed objects, comprising state information servers, and managed objects, which have input states that are formed by output states of other managed objects, comprising state information clients;

a state distributor system;

means for registering one or more specified managed objects with said state distributor system so that said one or more specified managed objects will receive at least one specified output state as an input state, said one or more specified managed objects thereby each becoming a state information client;

means for responding to a change in an output state of a managed object and for sending a transition indicating the output state change to said state distributor system;

said state distributor system comprising means for identifying state information clients registered for an output state change and for notifying the identified state information clients of the output state change;

wherein managed objects which comprise state information servers need not have information concerning all state information clients that are registered to receive their output states, and wherein said hierarchy of managed objects may thus be easily modified.

entities within said platform, said entity comprising one or more of said resources and one or more objects within said call processing system, said OAM&P system comprising:

a hierarchy of managed objects, each managed object having input states, output states, and a defined logical relation among said input states and said output states;

means defining one or more objects templates, each defined object template having a defined logical relation among input states and output states, and thus representing a basic structure to be utilized in creating transient objects; and

means for instantiating one or more transient objects at run time of the OAM&P system as a function of the types and amount of each type of resources used by said platform.

17. A resource managing system for assigning resources for use in performing call processing by a call processing system, a resource being assigned in response to a request made by said call processing system for a capability, said resource managing system comprising:

means for receiving a request made by said call processing system for a specified capability;

means defining a plurality of logical resource objects, each logical resource object comprising means for translating generic resource commands made by said call processing system into a form compatible with actual resources coupled to said call processing system;

means for correlating which logical resource objects support each capability within a set of capabilities that may be requested by said call processing system; and

means for allocating a logical resource object that supports said specified capability.

18. The resource managing system according to claim 17, further comprising means for confirming said allocation to said call processing system.

19. The resource managing system according to claim 18, wherein said request represents a plurality of different capabilities required by said call processing system; and wherein

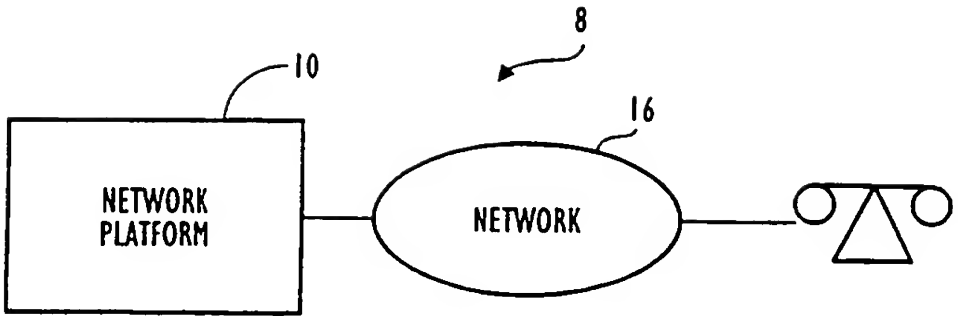

8
10
16
NETWORK PLATFORM
NETWORK

FIG. 1

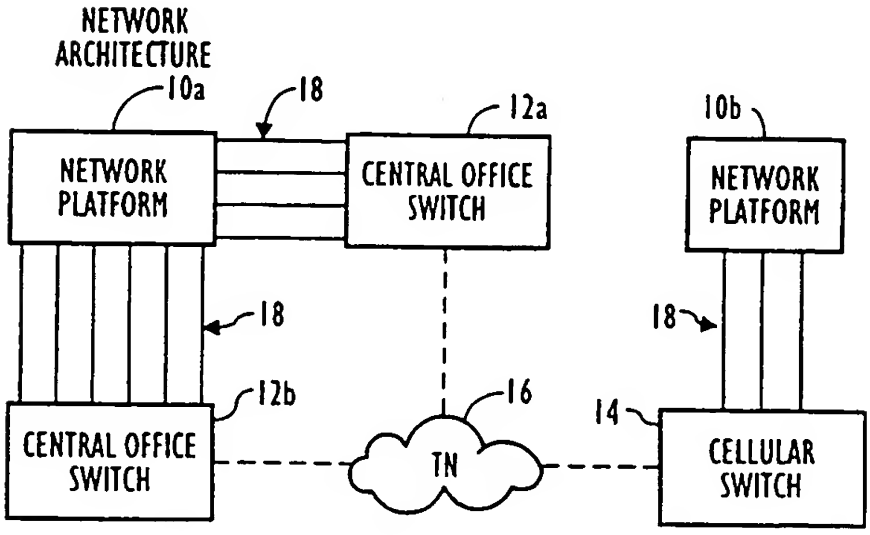

NETWORK ARCHITECTURE
10a
18
12a
10b
NETWORK PLATFORM
CENTRAL OFFICE SWITCH
NETWORK PLATFORM
18
18
12b
16
14
CENTRAL OFFICE SWITCH
TN
CELLULAR SWITCH

FIG. 2

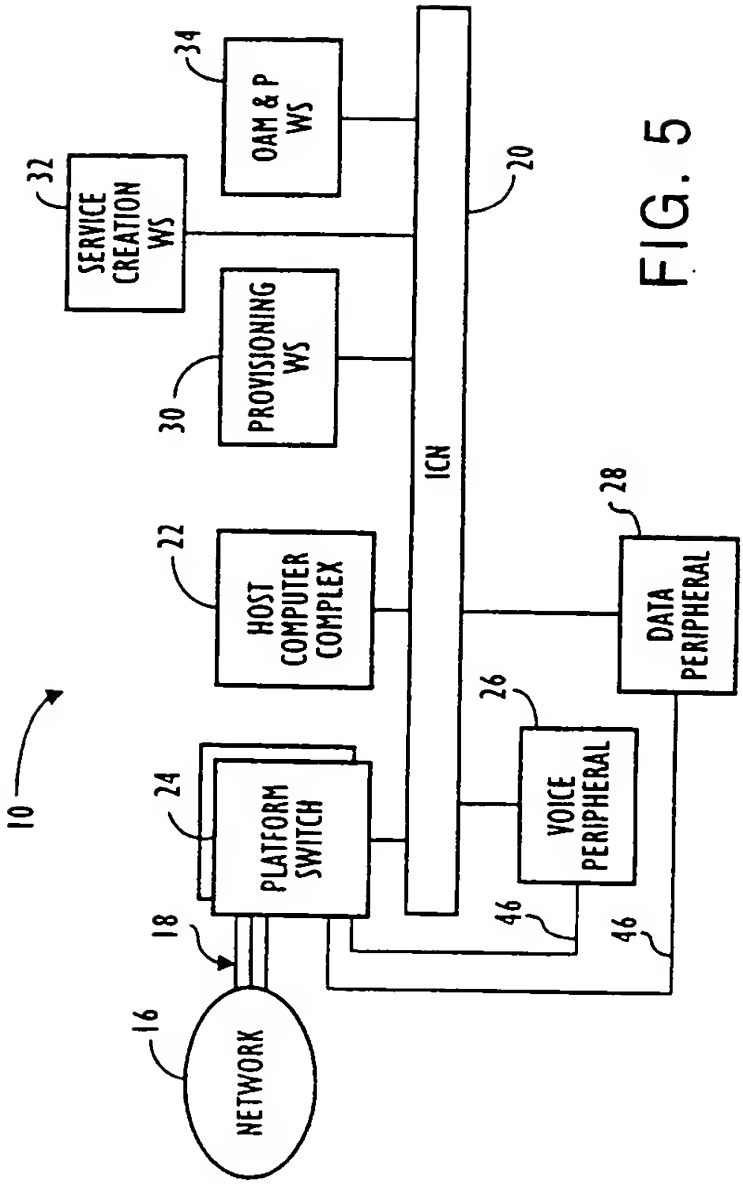

10
NETWORK
16
18
PLATFORM SWITCH
24
HOST COMPUTER COMPLEX
22
PROVISIONING WS
30
SERVICE CREATION WS
32
OAM & P WS
34
ICN
20
VOICE PERIPHERAL
26
DATA PERIPHERAL
28
46
46

FIG. 5

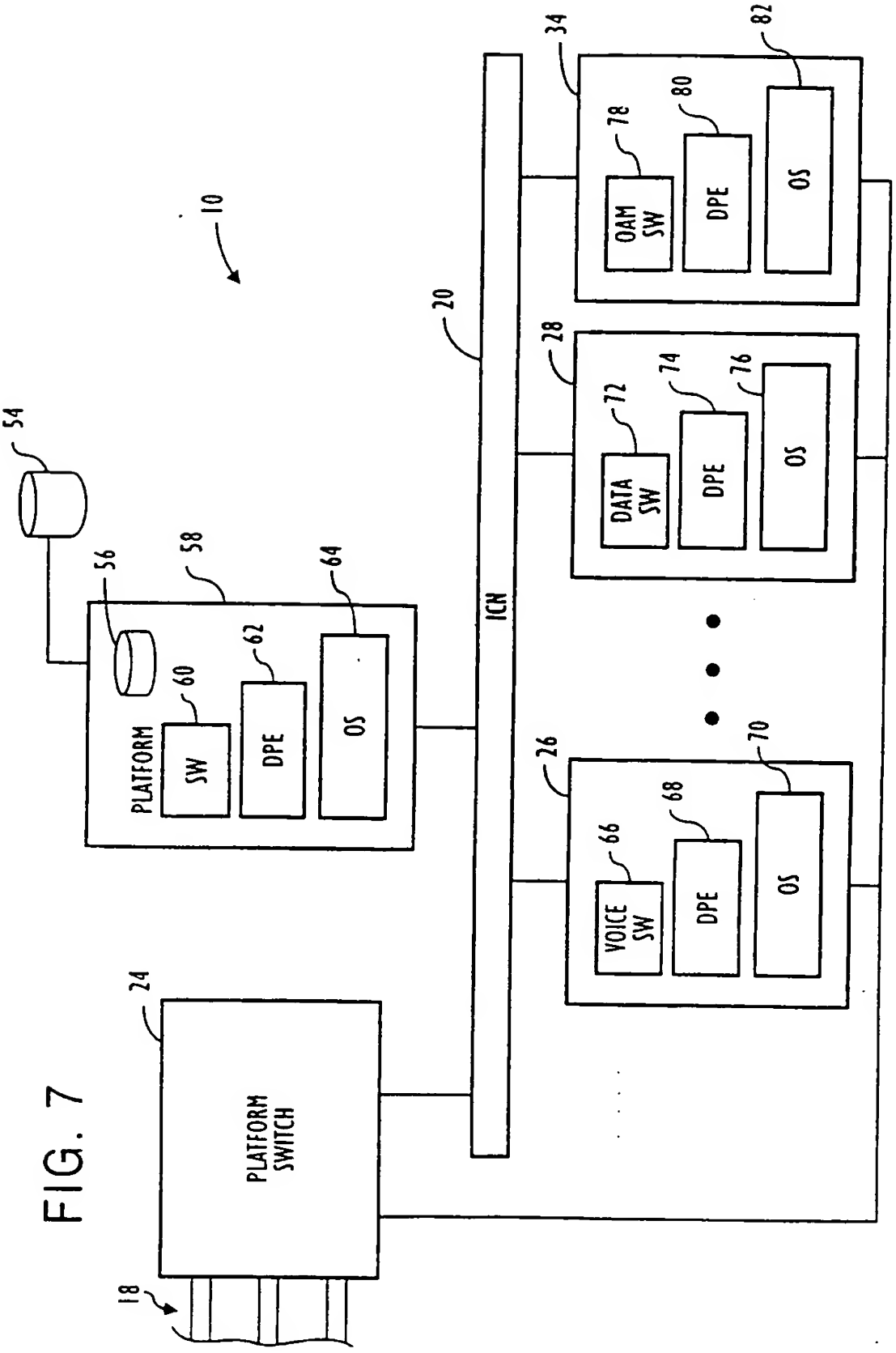

10
20
ICN
34
82
80
78
OAM SW
DPE
OS
28
74
76
72
DATA SW
DPE
OS
54
56
58
64
PLATFORM
60
SW
62
DPE
OS
26
70
68
66
VOICE SW
DPE
OS
24
PLATFORM SWITCH
18

FIG. 7

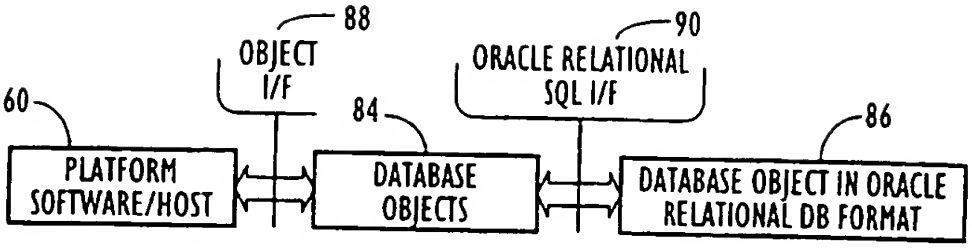


FIG. 9

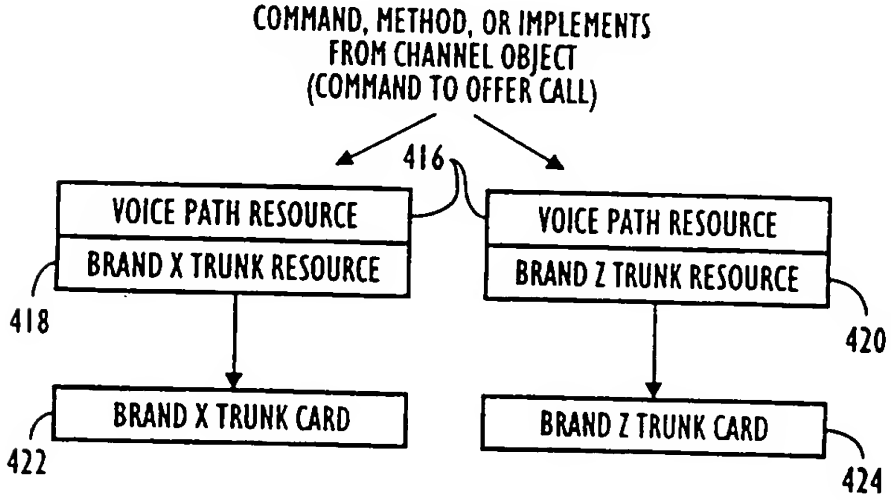


FIG. 10A

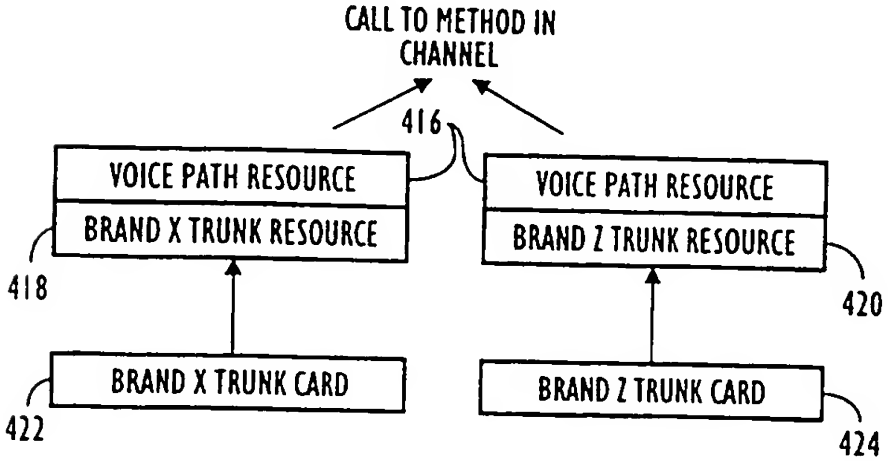


FIG. 10B

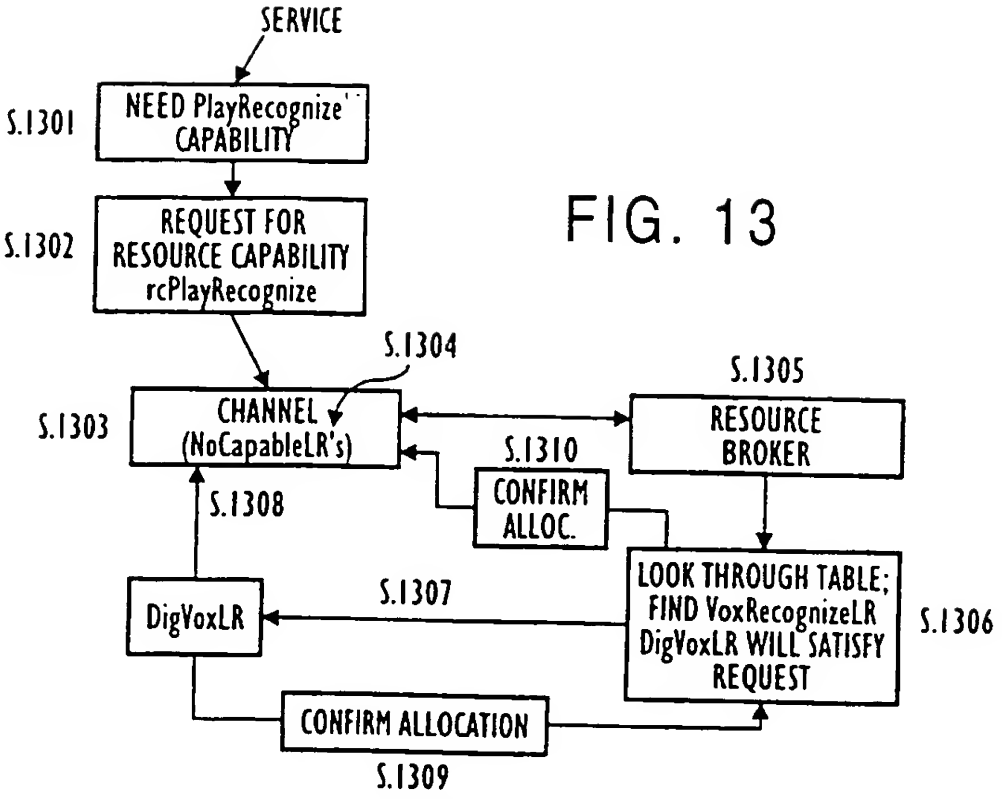

SERVICE
S.1301
NEED PlayRecognize CAPABILITY
S.1302
REQUEST FOR RESOURCE CAPABILITY rcPlayRecognize
S.1304
S.1305
S.1303
CHANNEL (NoCapableLR's)
RESOURCE BROKER
S.1310
CONFIRM ALLOC.
S.1308
LOOK THROUGH TABLE; FIND VoxRecognizeLR DigVoxLR WILL SATISFY REQUEST
S.1307
S.1306
DigVoxLR
CONFIRM ALLOCATION
S.1309
RESOURCE MANAGEMENT

FIG. 13

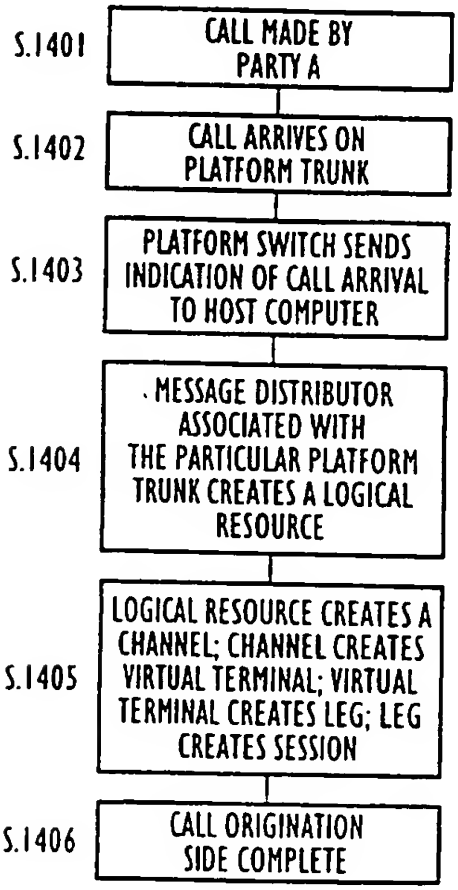

S.1401
CALL MADE BY PARTY A
S.1402
CALL ARRIVES ON PLATFORM TRUNK
S.1403
PLATFORM SWITCH SENDS INDICATION OF CALL ARRIVAL TO HOST COMPUTER
S.1404
MESSAGE DISTRIBUTOR ASSOCIATED WITH THE PARTICULAR PLATFORM TRUNK CREATES A LOGICAL RESOURCE
S.1405
LOGICAL RESOURCE CREATES A CHANNEL; CHANNEL CREATES VIRTUAL TERMINAL; VIRTUAL TERMINAL CREATES LEG; LEG CREATES SESSION
S.1406
CALL ORIGINATION SIDE COMPLETE

FIG. 14

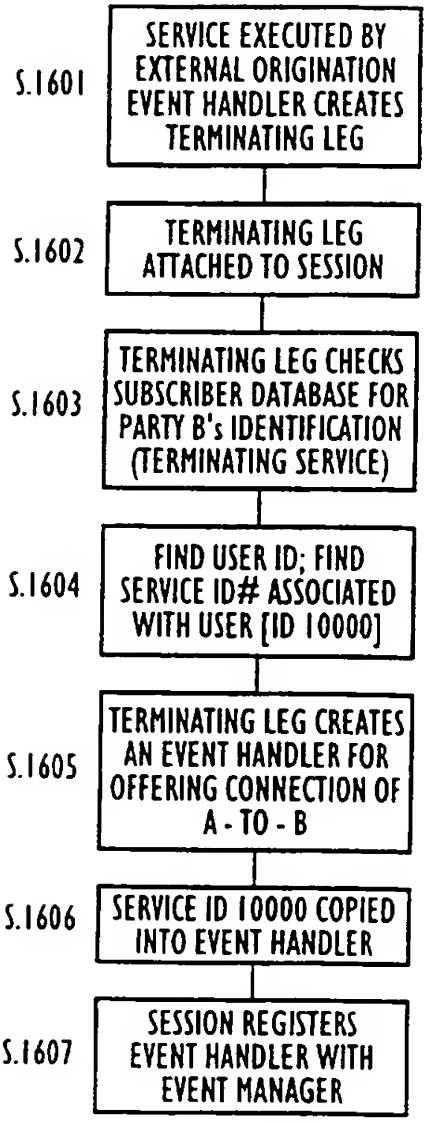

S.1601
SERVICE EXECUTED BY EXTERNAL ORIGINATION EVENT HANDLER CREATES TERMINATING LEG
S.1602
TERMINATING LEG ATTACHED TO SESSION
S.1603
TERMINATING LEG CHECKS SUBSCRIBER DATABASE FOR PARTY B's IDENTIFICATION (TERMINATING SERVICE)
S.1604
FIND USER ID; FIND SERVICE ID# ASSOCIATED WITH USER [ID 10000]
S.1605
TERMINATING LEG CREATES AN EVENT HANDLER FOR OFFERING CONNECTION OF A - TO - B
S.1606
SERVICE ID 10000 COPIED INTO EVENT HANDLER
S.1607
SESSION REGISTERS EVENT HANDLER WITH EVENT MANAGER

FIG. 16

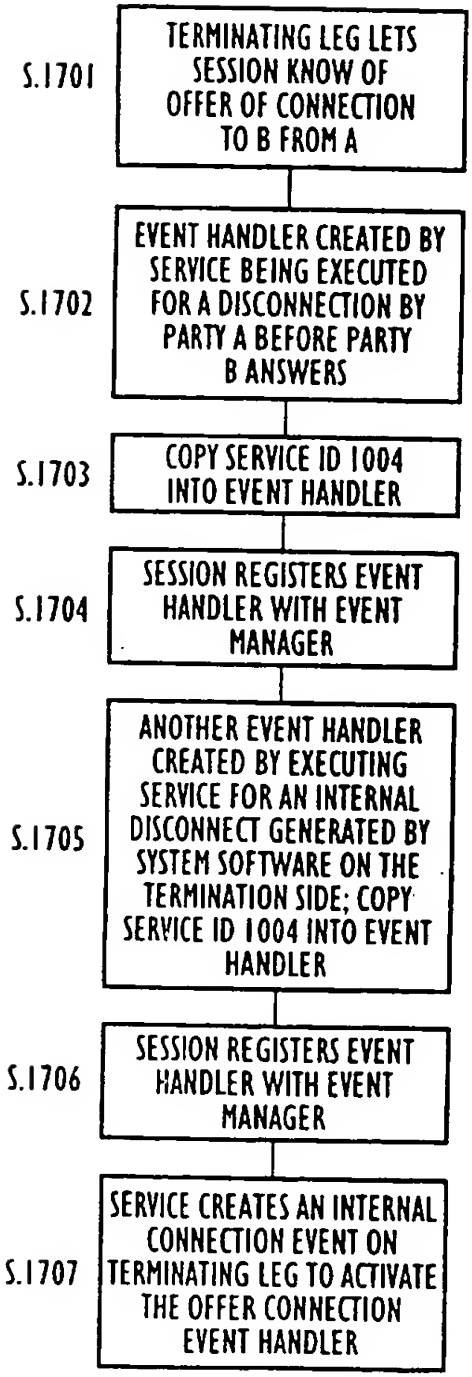

S.1701
TERMINATING LEG LETS SESSION KNOW OF OFFER OF CONNECTION TO B FROM A
S.1702
EVENT HANDLER CREATED BY SERVICE BEING EXECUTED FOR A DISCONNECTION BY PARTY A BEFORE PARTY B ANSWERS
S.1703
COPY SERVICE ID 1004 INTO EVENT HANDLER
S.1704
SESSION REGISTERS EVENT HANDLER WITH EVENT MANAGER
S.1705
ANOTHER EVENT HANDLER CREATED BY EXECUTING SERVICE FOR AN INTERNAL DISCONNECT GENERATED BY SYSTEM SOFTWARE ON THE TERMINATION SIDE; COPY SERVICE ID 1004 INTO EVENT HANDLER
S.1706
SESSION REGISTERS EVENT HANDLER WITH EVENT MANAGER
S.1707
SERVICE CREATES AN INTERNAL CONNECTION EVENT ON TERMINATING LEG TO ACTIVATE THE OFFER CONNECTION EVENT HANDLER

FIG. 17

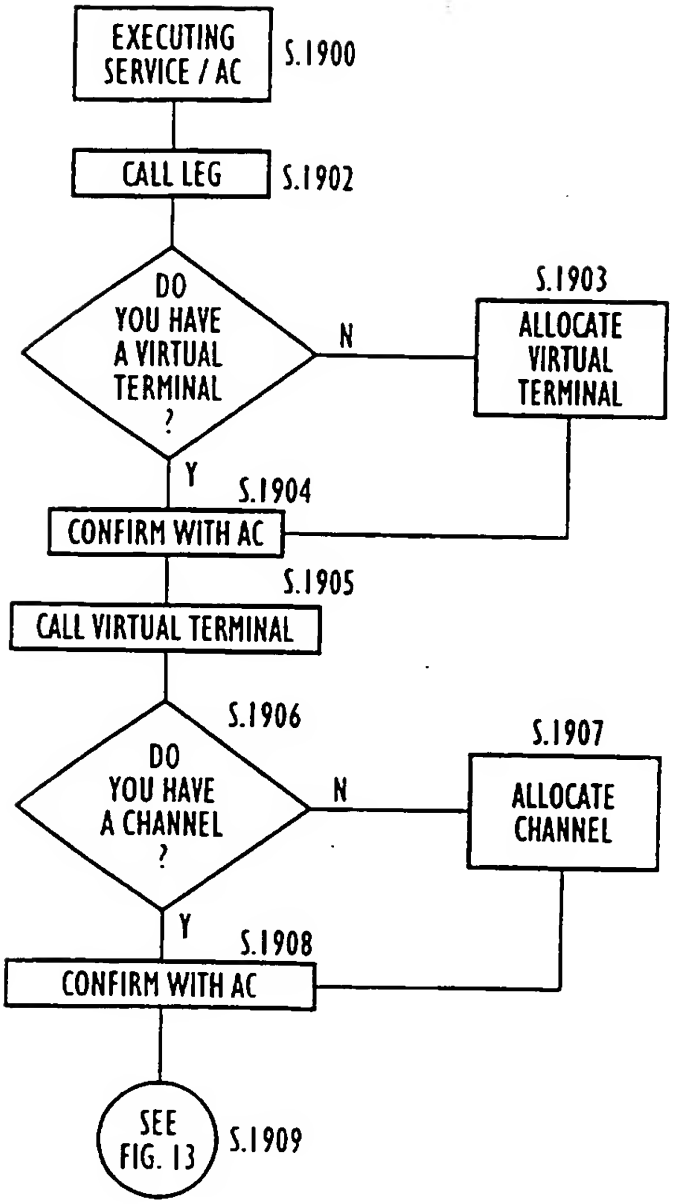

EXECUTING SERVICE / AC
S.1900
CALL LEG
S.1902
DO YOU HAVE A VIRTUAL TERMINAL ?
N
S.1903
ALLOCATE VIRTUAL TERMINAL
Y
S.1904
CONFIRM WITH AC
S.1905
CALL VIRTUAL TERMINAL
S.1906
DO YOU HAVE A CHANNEL ?
N
S.1907
ALLOCATE CHANNEL
Y
S.1908
CONFIRM WITH AC
SEE FIG. 13
S.1909

FIG. 19

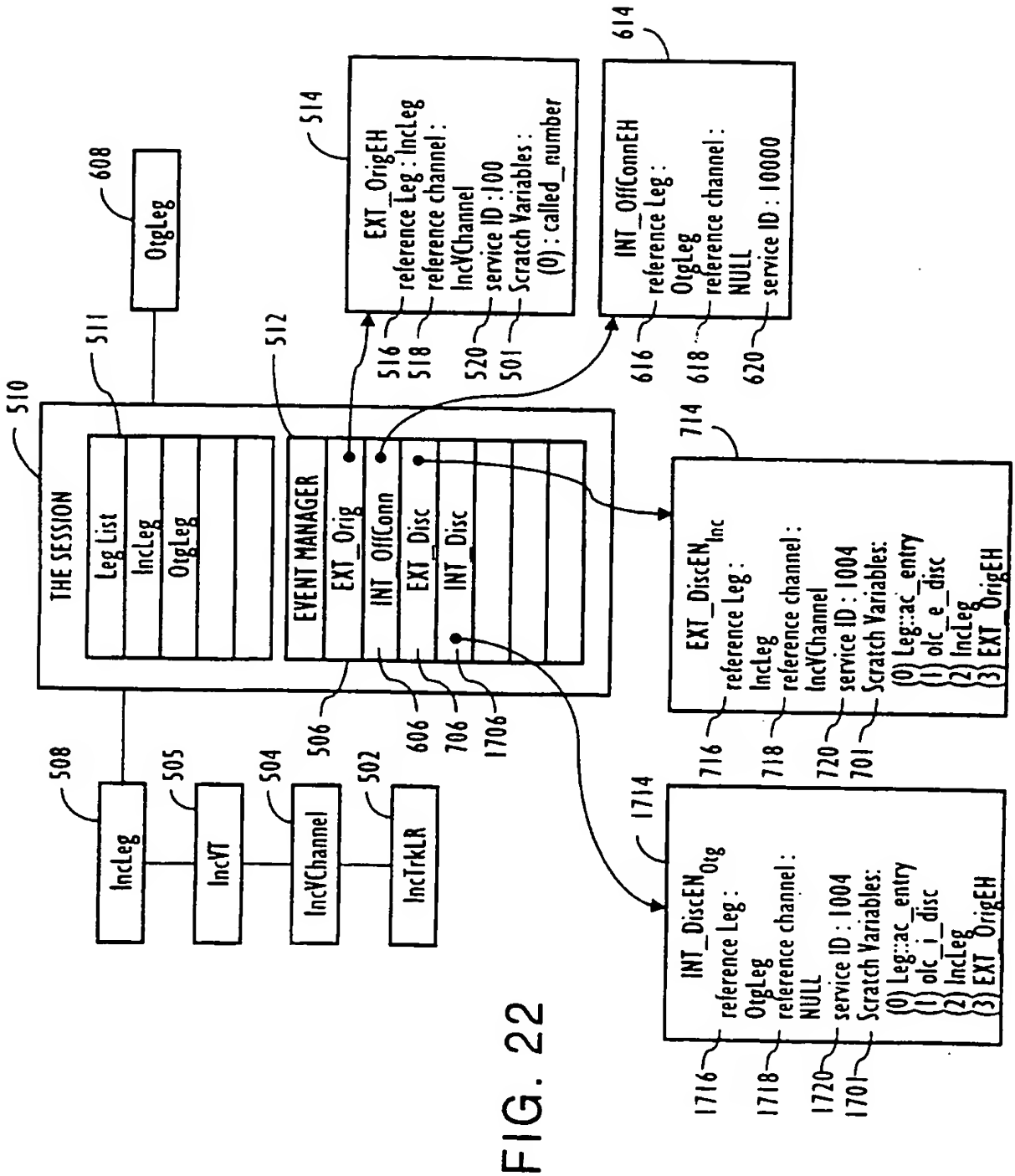

THE SESSION
Leg List
IncLeg
OtgLeg
EVENT MANAGER
EXT_Orig
INT_OffConn
EXT_Disc
INT_Disc
EXT_OrigEH
reference Leg : IncLeg
reference channel :
IncVChannel
service ID : 100
Scratch Variables :
(0) : called_number
INT_OffConnEH
reference Leg :
OtgLeg
reference channel :
NULL
service ID : 10000
EXT_DiscEN Inc
reference Leg :
IncLeg
reference channel :
IncVChannel
service ID : 1004
Scratch Variables:
(0) Leg::ac_entry
(1) olc_e_disc
(2) IncLeg
(3) EXT_OrigEH
INT_DiscEN Otg
reference Leg :
OtgLeg
reference channel :
NULL
service ID : 1004
Scratch Variables:
(0) Leg::ac_entry
(1) olc_i_disc
(2) IncLeg
(3) EXT_OrigEH
IncLeg
IncVT
IncVChannel
IncTrkLR
OtgLeg
510
511
512
514
516
518
520
501
608
614
616
618
620
714
716
718
720
701
1714
1716
1718
1720
1701
508
505
504
502
506
606
706
1706

FIG. 22

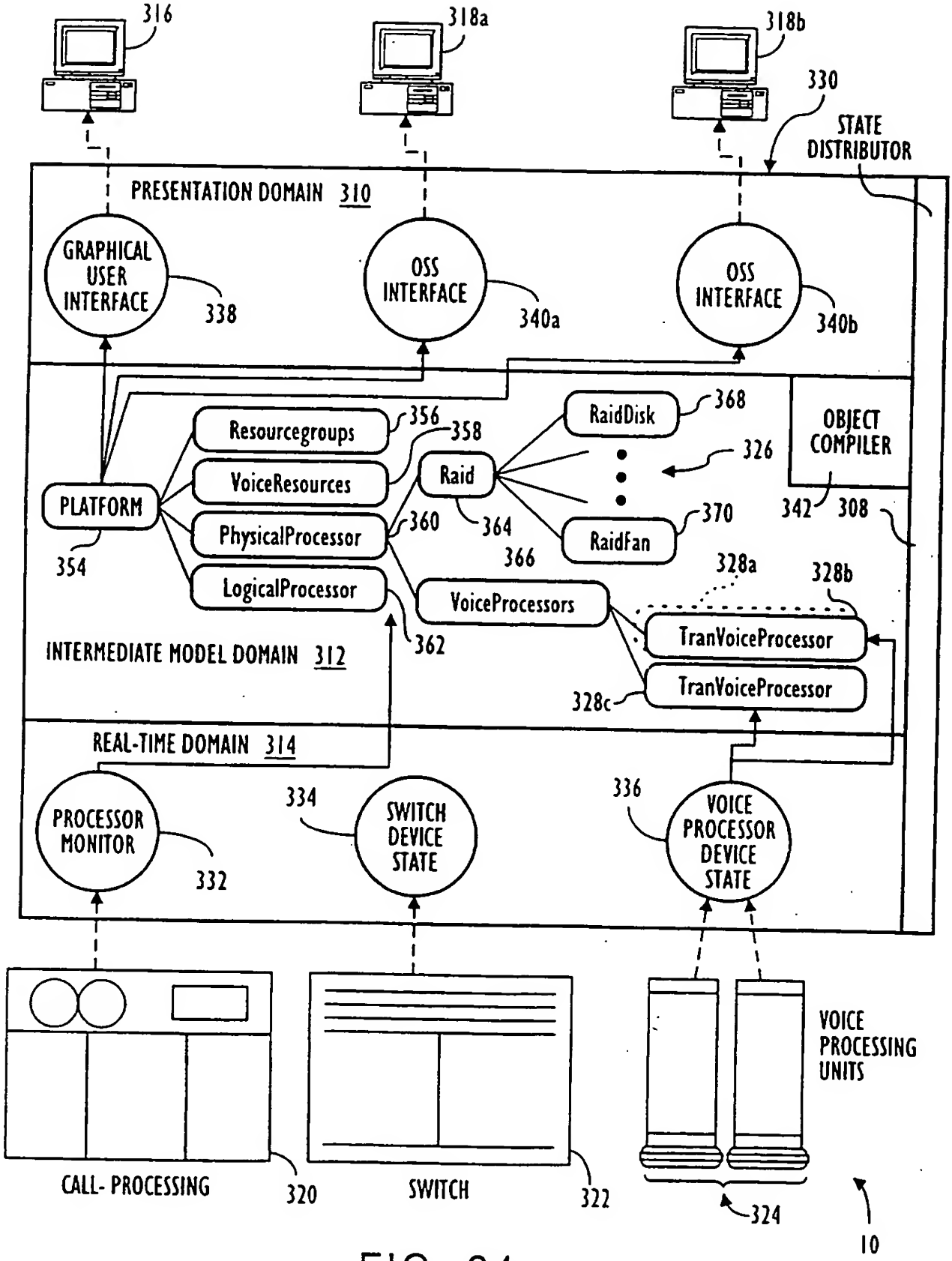

316
318a
318b
330
STATE DISTRIBUTOR
PRESENTATION DOMAIN 310
GRAPHICAL USER INTERFACE
338
OSS INTERFACE
340a
OSS INTERFACE
340b
OBJECT COMPILER
342
308
Resourcegroups
356
358
VoiceResources
PLATFORM
354
PhysicalProcessor
360
LogicalProcessor
362
Raid
364
366
RaidDisk
368
326
RaidFan
370
VoiceProcessors
328a
328b
TranVoiceProcessor
TranVoiceProcessor
328c
INTERMEDIATE MODEL DOMAIN 312
REAL-TIME DOMAIN 314
PROCESSOR MONITOR
332
334
SWITCH DEVICE STATE
336
VOICE PROCESSOR DEVICE STATE
CALL- PROCESSING
320
SWITCH
322
VOICE PROCESSING UNITS
324
10

FIG. 24

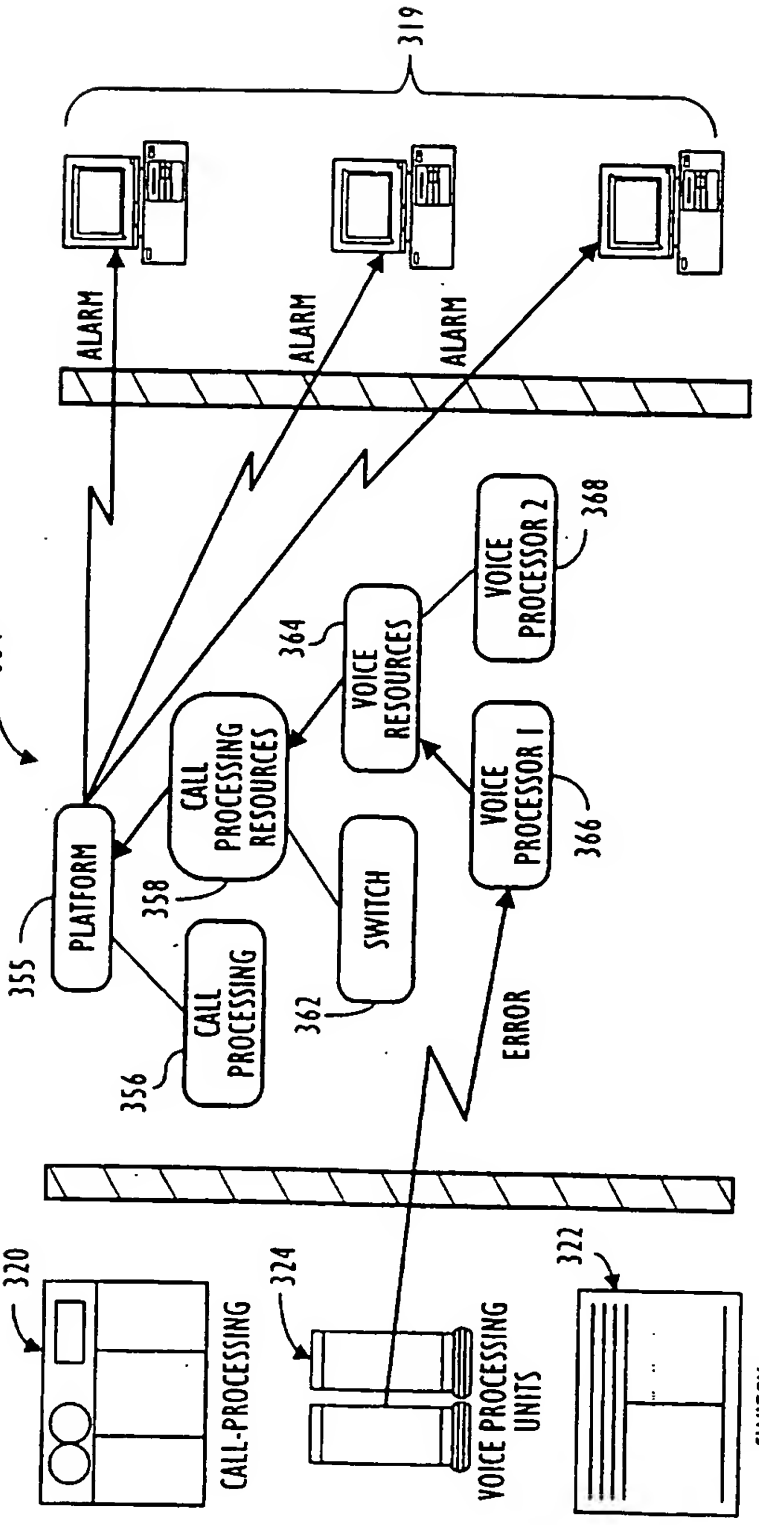
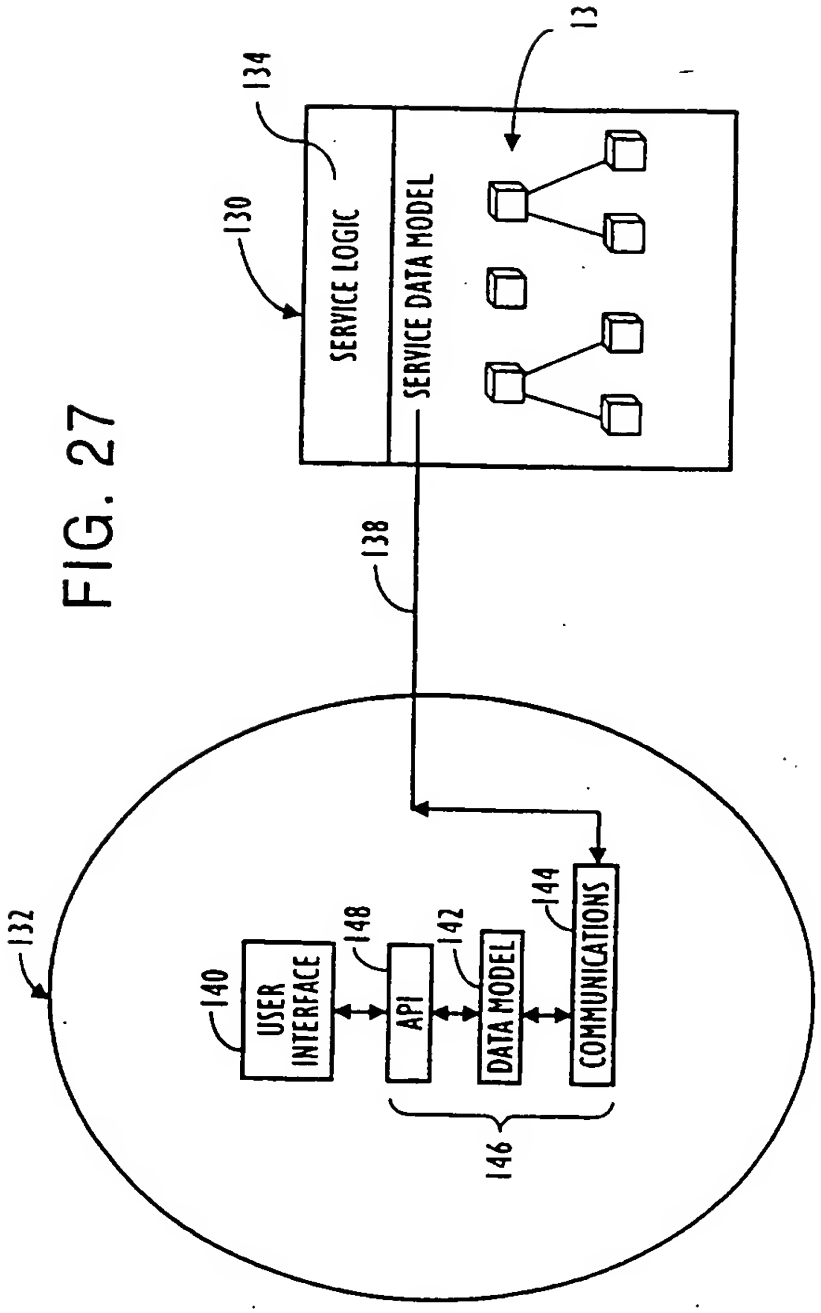


FIG. 25

FIG. 27


13
134
130
SERVICE LOGIC
SERVICE DATA MODEL
138
132
140
USER INTERFACE
148
API
142
DATA MODEL
144
COMMUNICATIONS
146

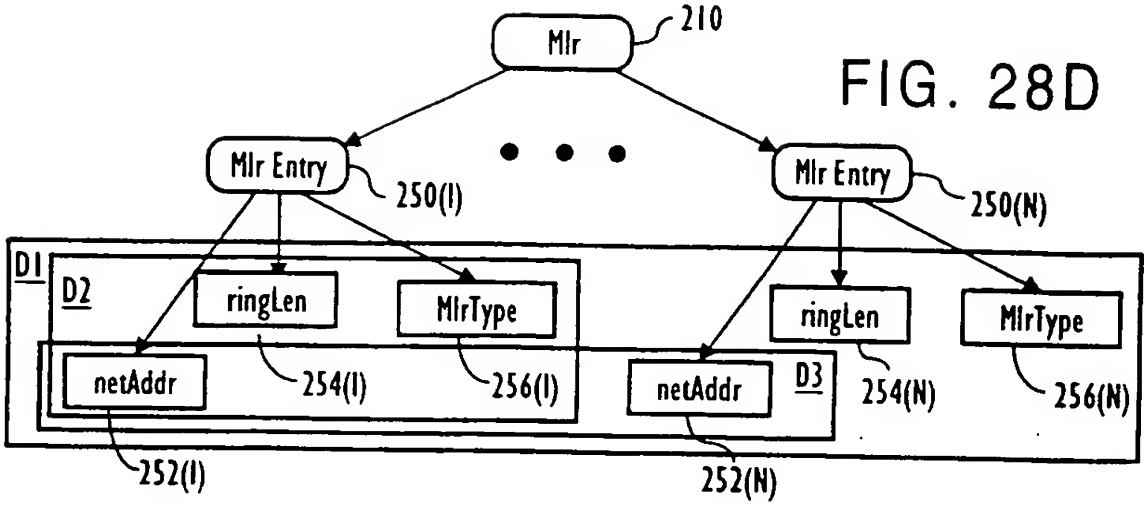


FIG. 28D

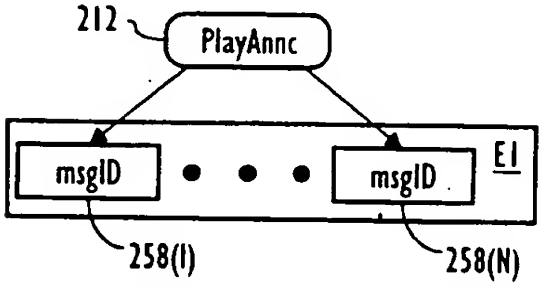


FIG. 28E

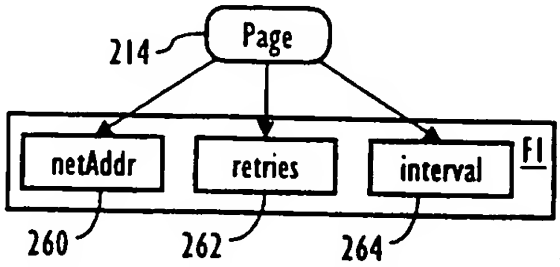


FIG. 28F

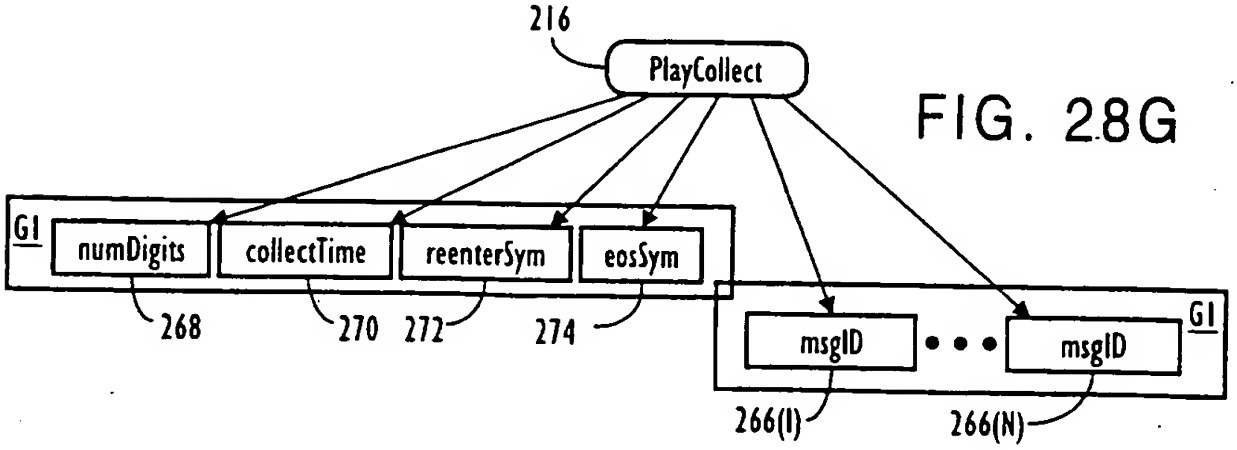


FIG. 28G

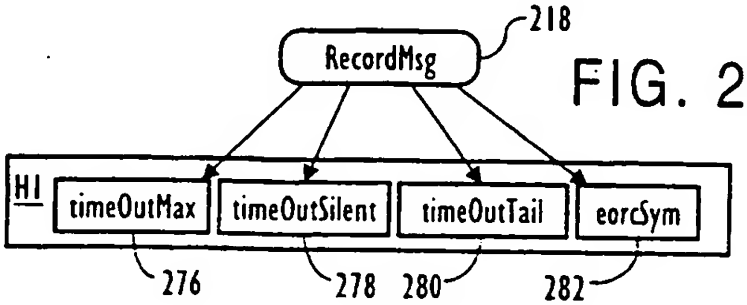


FIG. 28H

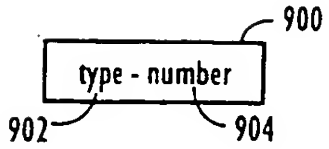


FIG. 29A

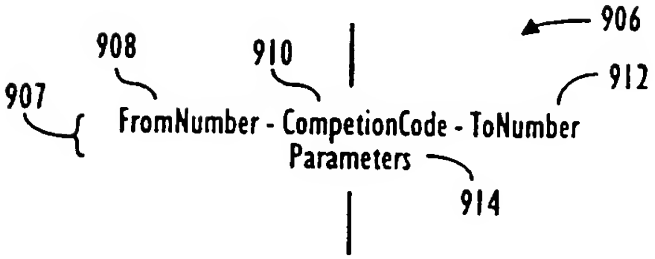


FIG. 29B

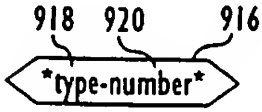


FIG. 29C

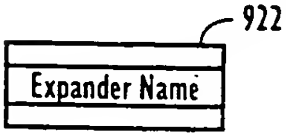


FIG. 29D

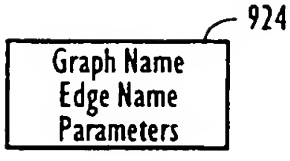


FIG. 29E

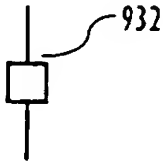
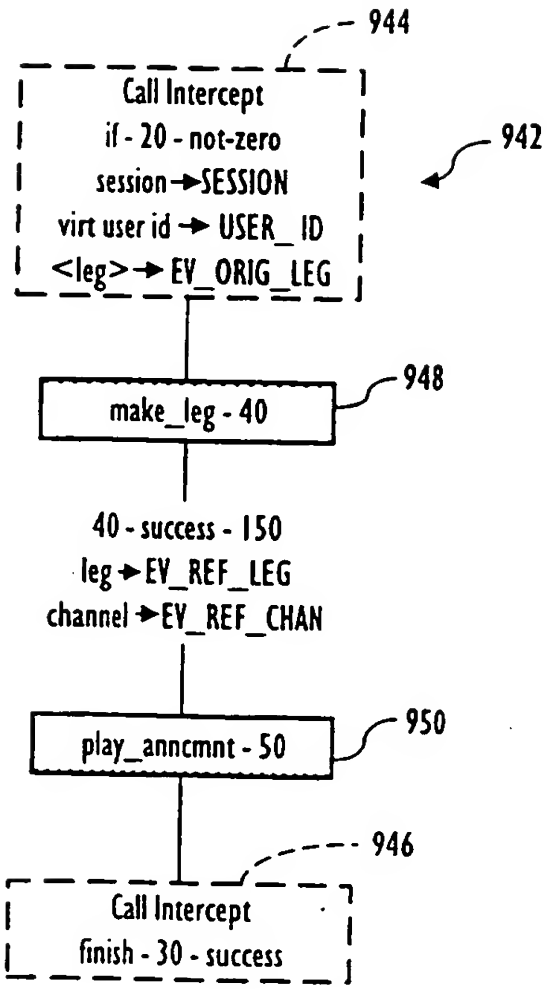


FIG. 29F

FIG. 31


944
Call Intercept
if - 20 - not-zero
session → SESSION
virt user id → USER_ID
<leg> → EV_ORIG_LEG
942
make_leg - 40
948
40 - success - 150
leg → EV_REF_LEG
channel → EV_REF_CHAN
play_anncmnt - 50
950
946
Call Intercept
finish - 30 - success

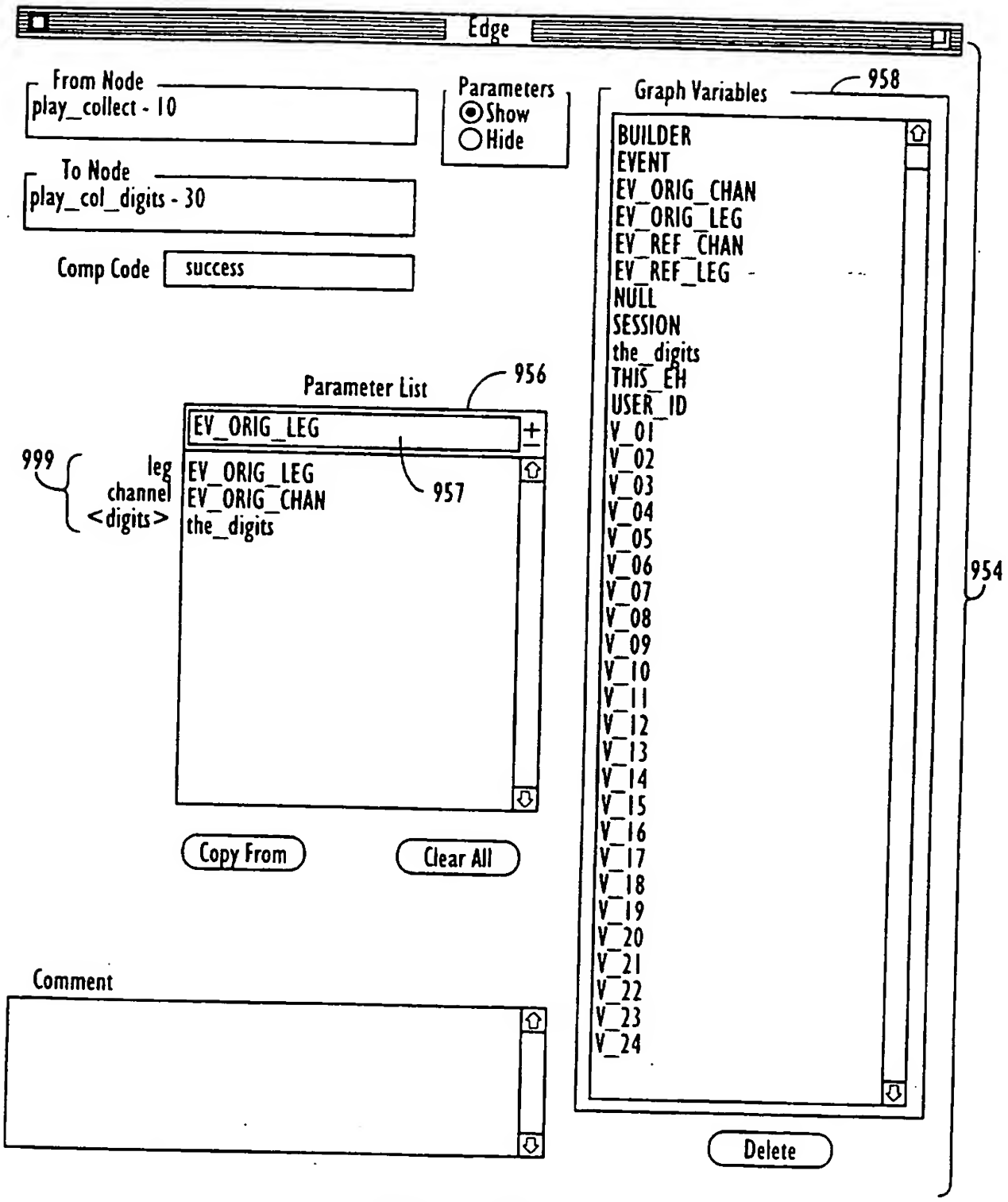

Edge
From Node
play_collect - 10
To Node
play_col_digits - 30
Comp Code
success
Parameters
Show
Hide
Graph Variables
958
BUILDER
EVENT
EV_ORIG_CHAN
EV_ORIG_LEG
EV_REF_CHAN
EV_REF_LEG
NULL
SESSION
the_digits
THIS_EH
USER_ID
V_01
V_02
V_03
V_04
V_05
V_06
V_07
V_08
V_09
V_10
V_11
V_12
V_13
V_14
V_15
V_16
V_17
V_18
V_19
V_20
V_21
V_22
V_23
V_24
954
Parameter List
956
EV_ORIG_LEG
999
leg
channel
<digits>
EV_ORIG_LEG
EV_ORIG_CHAN
the_digits
957
Copy From
Clear All
Delete
Comment

FIG. 33

Box III TEXT OF THE ABSTRACT (Continuation of item 5 of the first sheet)

The technical features mentioned in the abstract do not include a reference sign between parentheses (PCT Rule 8.1(d)).

NEW ABSTRACT

A flexible network platform (10a) and call processing system are disclosed. The call processing system includes a particular call processing architecture and a resource managing system. An OAM & P subsystem (94) and systems for performing data provisioning (92) and service creation are each provided. The call processing system may include a call processing mechanism for performing call processing in accordance with defined service logic, and call processing resources connected to the call processing mechanism. The call processing resources are designed so that any service logic unit may be linked to any particular event that might occur in connection with the call processing system. A resource managing system (428) is also disclosed for assigning resources in respond to a request made by the call processing system specified capability. The resource managing system receives the request for a specified capability and allocates a logical resource object that identifies a particular resource that will support the specified resource capability.